GENERALISED ADDITIVE MIXED MODELS FOR DYNAMIC ANALYSIS IN LINGUISTICS: A PRACTICAL INTRODUCTION[1]

Márton Sóskuthy
University of York

[last updated: March 17, 2017]

1 Introduction

This is a hands-on introduction to Generalised Additive Mixed Models (GAMMs; Wood 2006) in the context of linguistics with a particular focus on dynamic speech analysis. Dynamic speech analysis is a term used to refer to analyses that look at measureable quantities of speech that vary in space and/or time. Temporal variation can be short-term (e.g. formant contours and pitch tracks) or long-term (e.g. diachronic change or change over the life-span). Similarly, spatial variation can sometimes be measured in millimetres (e.g. tongue contours), and sometimes in kilometres (e.g. the acoustic realisation of a sound category as a function of location on a dialect map). The focus of this introduction is mostly on short-term temporal variation in phonetics, and more specifically on formant trajectories (though one of the examples also involves diachronic change). This choice is mostly practical: I chose to illustrate GAMMs through formant trajectories simply because I'm comfortable talking about them. However, most of the concepts and techniques discussed here are more general and are also applicable to other types of short-term and long-term temporal trajectories.

The main goal of this introduction is to explain some of the main ideas underlying GAMMs, and to provide a practical guide to frequentist significance testing using these models. Some of the suggestions below are based on simulation-based work presented in Sóskuthy (2016) and Sóskuthy (in prep), but this introduction is meant as a standalone guide.

The following discussion is divided into two parts, which can be read in slightly different ways. The first part (section 2) looks at what GAMMs actually are, how they work and why/when we should use them. Although the reader can replicate some of the example analyses in this section, this is not essential – reading the section should be enough. The second part (section 3) is a tutorial introduction that illustrates the process of fitting and evaluating GAMMs in the R statistical software environment (R Core Team, 2013), and the reader is strongly encouraged to work through the examples on their own machine.

Since a lot of the research in GAMM theory is closely intertwined with software development in R (e.g. the author of one of the main textbooks on GAMM, Wood 2006 is also the maintainer of one of the main GAMM software packages), it is difficult to talk about GAMMs without using some of the terminology and conventions of R. Therefore, although the discussion in the first part is mostly conceptual, I do rely on R code to illustrate the structure of different models. However, I've tried not to use too much code, and it should be possible to follow the discussion without a strong background in R. The second part relies much more heavily on R and will be mostly of interest to readers who

[1] This introduction has been updated a few times. Thanks to Martijn Wieling, Bodo Winter and Donald Derrick for helpful comments and suggestions. Details of the changes are shown on the associated GitHub page (see end of section 1 for the link).

want to fit their own models using R. Readers who want to learn more about R before reading this introduction may want to consult Baayen (2008) and Johnson (2008), who both provide thorough introductions to R for beginners using examples from linguistics. GAMMs are a type of regression model and they are closely related to mixed effects regression. This tutorial assumes some background in regression modelling and it will help to be familiar with mixed effects models as well. Winter (2013) is a short but excellent introduction to mixed effects modelling, while Gelman & Hill (2007) and Baayen (2008) provide more in-depth treatments of the topic.

The examples in this introduction rely on two R packages: `mgcv` (Wood, 2006) and `itsadug` (van Rij et al., 2016). These should be installed and loaded before trying to run the analyses that follow. I've also put together a script with a few 'GAMM hacks' (`gamm_hacks.r`), which help to keep the code in the tutorial tidier and may also be useful for the reader's own analyses. This should be sourced after loading `itsadug`, as it overrides some of the functions in that package. Finally, the data sets for the tutorial are in two separate files called `words_50.csv` and `glasgow_r.csv`, which should be imported into R as `words.50` and `gl.r`.

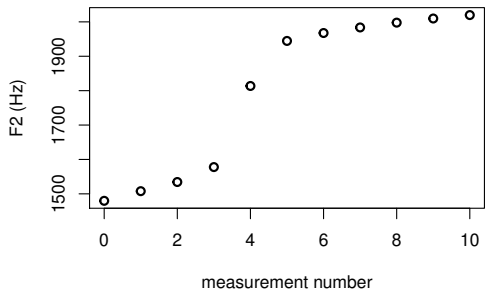
The data sets, the GAMM hacks script, the slides for Sóskuthy (2016) and the source code for the PDF are all available from my GitHub page:
https://github.com/soskuthy/gamm_intro.

I relied on a number of sources for this introduction, but the main body of the text is light on references to make the discussion easier to follow. There is a more detailed list in the final section that also includes links and brief summaries of each source.

2 A gentle introduction to GAMM theory

2.1 GAMs

Before discussing GAMMs, let's start with a slightly simpler type of model called Generalised Additive Models (that's GAMM without the 'mixed' part). The easiest way to understand GAMs is through a comparison with linear regression models. We will work through a specific example, where our goal will be simply to fit a regression line/curve to an F2 trajectory. The formant measurements are in Hz and the trajectory is represented by 11 measurement points taken at equal intervals (going from the very beginning to the very end). The figure below shows the trajectory (available as `traj.csv` from the GitHub page):



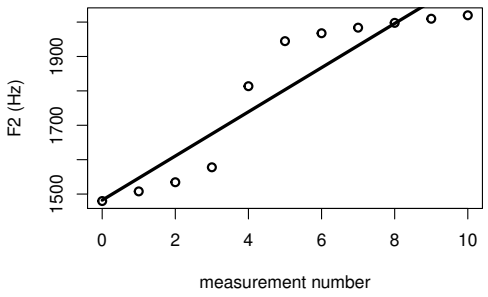
Here is a linear model that fits a line to the trajectory in R:

```
traj <- read.csv("traj.csv")  # importing the data
demo.lm <- lm(f2 ~ measurement.no, data = traj)
```

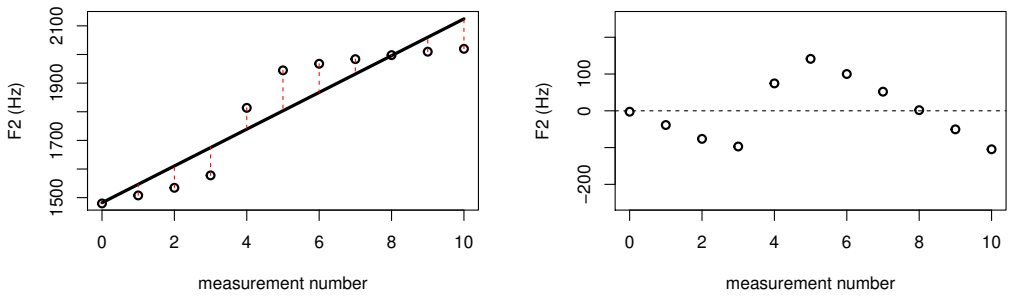
Though I'll try to use mathematical notation sparingly in this tutorial, it is useful to write out the model as a formula, as it will help with the transition to GAMs:[2]

$$Y_{\text{f2}} = \alpha + \beta_1 X_{\text{measurement.no}} \tag{1}$$

This is simply an equation for a straight line: $X_{\text{measurement.no}}$ stands for the values along the x-axis (measurement number), while Y_{f2} stands for corresponding values along the y-axis (F2 in Hz). α and β_1 specify the *intercept* (i.e. the height) and the *slope* of the regression line that represents the relationship between F2 and `measurement.no`. When we fit a regression line to the actual trajectory, the result looks like this:



The regression line is unable to capture the non-linear nature of the trajectory. This is bad news for models that seek to make causal inferences about such trajectories. Such discrepancies between the data and the model create systematic patterns in the *residuals* of the model (i.e. the difference between the predicted vs. the actual value of the outcome variable; in this case, the distances between the prediction line and the individual data points along the y-axis), which makes confidence intervals and p-values unreliable. This is illustrated by the following two figures. The figure on the left shows how the residuals are calculated by explicitly indicating the distance between the predicted and the observed values of the outcome variable. The figure on the right plots the raw residuals (the lengths of the red dashed lines on the left) against measurement number:



[2] I've left out the so-called error term to keep things simple, though this should technically be part of the model specification.

The residuals shouldn't vary systematically as a function of other predictors, yet there is a clear pattern in the figure above, which – in this case – results from the inability of the model to capture non-linearities. If we want to account for non-linear relationships, we'll need to take a different approach.

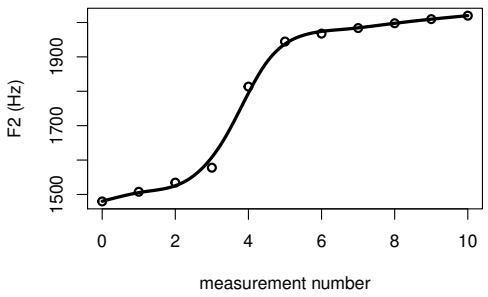
GAMs provide one way of getting around this problem. Let us refer to the slopes and intercepts of linear regression models as *parametric terms*. GAMs differ from traditional linear regression models by allowing so-called *smooth terms* alongside parametric terms. Smooth terms are extremely flexible, so much so that their mathematical representation in model specifications is simply 'some function of x':

$$Y_{\text{f2}} = \alpha + f_1(X_{\text{measurement.no}}) \tag{2}$$

This model specification does not say anything about the shape of the function linking $X_{\text{measurement.no}}$ and Y_{f2}. The only requirement on the smooth term $f_1(X_{\text{measurement.no}})$ is that it should be a smooth function of one or more predictor variables. Since the shape of this smooth function is not constrained in the same way as it is for regression lines, GAMs can deal with non-linearity. The following R function can be used to fit a GAM with the structure above to the F2 trajectory:[3]

```
demo.gam <- bam(f2 ~ s(measurement.no, bs = "cr"), data = traj)
```

The `s()` notation is used to distinguish smooth terms from parametric ones. The `bs="cr"` parameter tells R to use a so called 'cubic regression spline' as the smooth term (`bs` actually stands for 'basis', a concept that will be discussed in more detail below). The model fit is shown below.



So how are such wiggly smooth terms created in practice? We are not going to discuss the mathematical underpinnings of GAMs, but there are two fundamental and relatively straightforward concepts that need to be understood if one wants to work with these models: *basis functions* and the *smoothing parameter*. Basis functions are simple functions that add up to a (potentially) more complex curve. For instance, consider the following model:

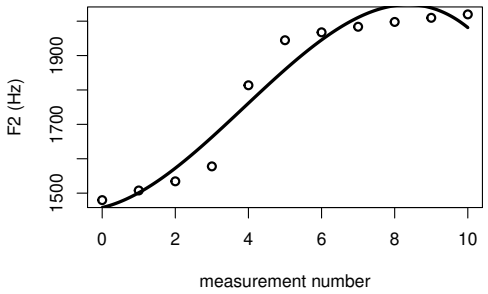
[3] Throughout this tutorial, we'll use the `bam()` function from the `mgcv` package to fit GAMs and GAMMs. An alternative is the `gam()` function from the same package, but `gam()` is less flexible and often slower than `bam()`, so we won't use it here.

```
demo.poly <- lm(f2 ~ measurement.no + I(measurement.no^2) +
                     I(measurement.no^3), data=traj)
```

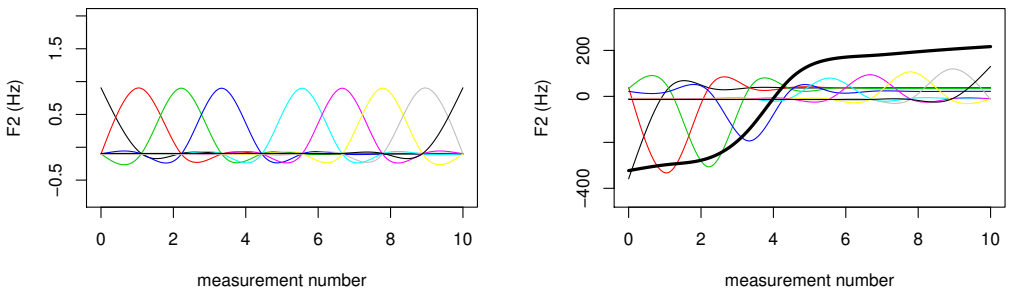
Or, in mathematical notation:

$$Y_{\text{f2}} = \alpha + \beta_1 X_{\text{measurement.no}} + \beta_2 X^2_{\text{measurement.no}} + \beta_3 X^3_{\text{measurement.no}} \tag{3}$$

And the model fit:



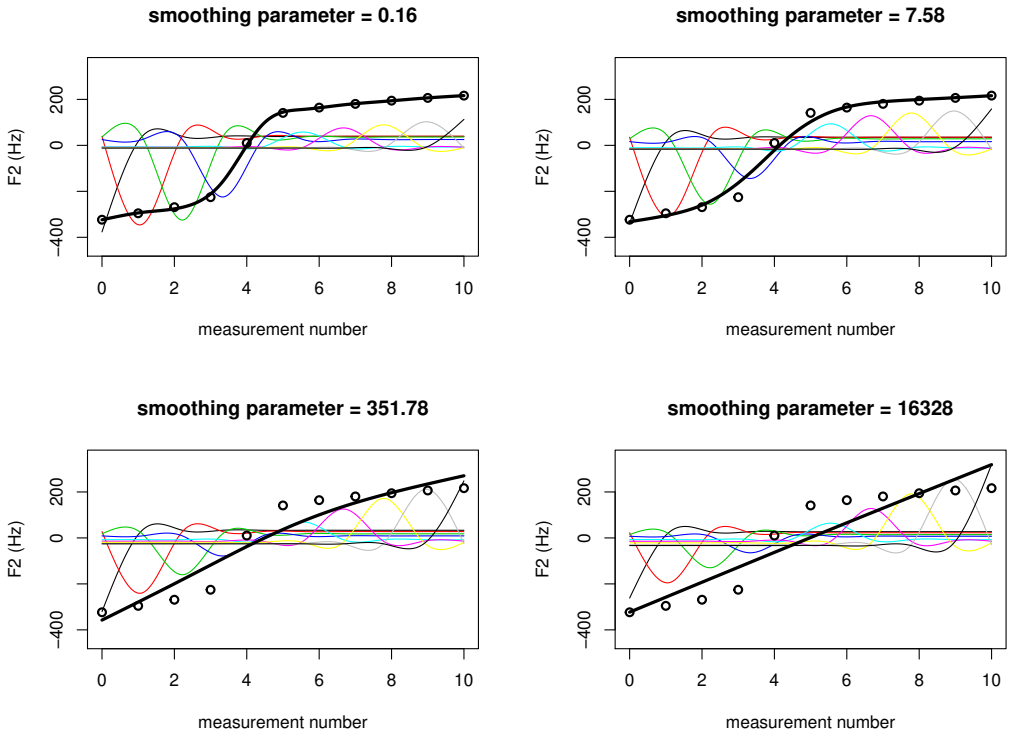
This is an example of polynomial regression, where both a variable x and some of its powers (x^2, x^3, ...) are included as predictors. The terms x, x^2, x^3 are referred to as basis functions. The fitted curve is obtained by multiplying each of the basis functions by the corresponding coefficient and then adding them up. The plot on the left below shows the basis functions of the GAM smooth for the formant trajectory before multiplication by the coefficients. The plot on the right shows the same basis functions after multiplication, and their sum (i.e. the predicted formant trajectory minus the intercept term).



The above smooth is made up of 9 basis functions, which is a default setting for certain types of smooths in R. Note that the basis functions are placed at regular intervals, and that they converge on each other at certain points (this is clearer in the plot on the left-hand side). These 'convergence points' are called *knots*, and there are 10 of them (the number of basis functions + 1): 2 at the edges of the plot and 8 in the middle. There is a simple intuition linked to the number of knots / basis functions: increasing this number allows for more wiggliness in the smooth, while decreasing it makes the smooth... well, smoother.

In linear regression models where the basis functions are included manually (e.g. polynomial regression), the number of basis functions (or knots) has to be chosen by the

modeller. This can be tricky, as using too few basis functions can lead to oversmoothing (i.e. missing some of the non-linearity in the data), while using too many can lead to overfitting (i.e. missing the real trend in the data by fitting a curve to random noise). This is where GAMs really shine. GAMs rely on a value called the smoothing parameter. The coefficients for the individual basis functions contained in a GAM smooth are estimated in such a way that the resulting curve has a controlled degree of wiggliness determined by the smoothing parameter. The higher the smoothing parameter, the smoother (= less wiggly) the estimated curve. This is illustrated below, where the number of basis functions is always the same, but the value of the smoothing parameter is varied.[4]



In other words, the degree of smoothness / wiggliness in GAMs is mostly determined by the smoothing parameter. The role of the number of basis functions / knots is (mostly) reduced to setting an upper limit on the degree of wiggliness: having 3 knots (= 2 basis functions) obviously allows for much less wiggliness than 10 or 100 knots, even if we set the smoothing parameter extremely low.

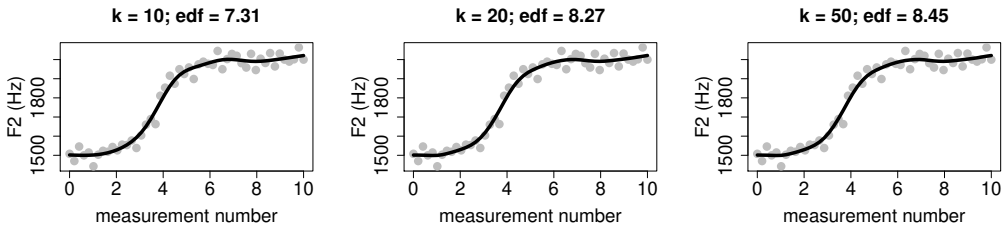
The GAMs that we use in this introduction (i.e. the ones implemented in the `mgcv` package) do not require the modeller to decide on a value for the smoothing parameter: they estimate it directly from the data. This can be done using methods based on cross-validation or maximum likelihood estimation. We won't discuss these in detail, but a brief overview will be helpful (see `?gam.selection` from `mgcv` for more detail and references). Both sets of methods aim at choosing a value for the smoothing parameter that makes the resulting curve generalisable beyond the specific sample under investigation. This is especially clear in the case of cross-validation. Cross-validation

[4] As before, the curve is shown without the intercept, which means that it is centred around 0 along the *y*-axis. This is why some of the values are negative. The graphs also include shifted versions of the actual data points to make the degree of smoothing clearer.

works by (i) creating subsets of the full data set that each exclude a single data point, (ii) refitting the model to each of these subsets and (iii) checking how well the fitted models predict the excluded data points. Choosing a value for the smoothing parameter that is too low will result in a curve that is overly wiggly. This excess wiggliness will be used to capture idiosyncratic variation in the data set, which leads to bad performance on out-of-sample observations (and therefore a high error-rate in cross-validation). If the smoothing parameter is too high, the curve will fail to capture non-linear patterns both in within-sample and out-of-sample observations, which, again, leads to a high error-rate in cross-validation. Note that `mgcv` actually only fits the model once, and uses a mathematical trick to calculate the cross-validation score (see Wood 2006 for more detail). Maximum likelihood estimation relies on a different technique that works by treating smooths as random effects for the purposes of estimating their smoothing parameters (but the outcome is the same: a curve that – all things being equal – avoids under/overfitting and is generalisable beyond the sample).

As a result of smoothing parameter estimation, the number of basis functions has little bearing on the shape of the smoother provided that there are enough of them to represent the degree of wiggliness in the data, so a smooth with a high number of basis functions will often look very similar to one with a lower number. This is illustrated below using a slightly modified version of our formant trajectory with 50 measurement points instead of 11 and a small amount of added noise. Three models were fit to this trajectory with different values for `k` (10, 20 and 50). The plots show the data, the fitted curves and the so-called *estimated degrees of freedom*, or EDF (see below). All three curves look very similar.

```
demo.gam.k.10 <- bam(f2 ~ s(measurement.no, bs = "cr", k = 10),
                     data = traj.50)
demo.gam.k.20 <- bam(f2 ~ s(measurement.no, bs = "cr", k = 20),
                     data = traj.50)
demo.gam.k.50 <- bam(f2 ~ s(measurement.no, bs = "cr", k = 50),
                     data = traj.50)
```



On the basis of the above discussion, it would seem that the best strategy is to set `k` to a high value so that model is not restricted in terms of how much wiggliness it can attribute to the underlying curve. And once we've done that, we could forget about the whole thing, basically trusting `mgcv` to do the right thing for us.

Unfortunately, that's not a good strategy. As is often the case with statistical methods, knowing only a little about GAMs and setting up models without understanding what they do is probably worse than using less advanced methods in an informed way. First of all, although cross-validation and other estimation methods attempt to avoid overfitting, they are not always successful at doing so, which means that an unrealistically high value of `k` does sometimes lead to an unrealistically wiggly curve. Second, there are

several situations where the number of basis functions needs to be changed, and these are almost impossible to avoid while working with dynamic speech data:

- *too few measurements to support* `k` *knots*: If our trajectories only have 11 measurements, the maximum number of knots is also 11 (that is, the maximum number of basis functions is 10). Since the default value for `k` is 10 (although this number may be different depending on the type of smooth), it needs to be lowered if there are less than 10 unique values for a given variable.

- *not enough wiggliness allowed*: The default value of `k` can only support a certain amount of wiggliness in the data. If the actual trajectories show a greater degree of non-linearity, `k` needs to be increased.

- *computational inefficiency due to high* `k`: The higher the value of `k`, the longer it will take to fit the model. Therefore, it is a good idea not to increase `k` any further than necessary. In realistic scenarios, the modeller may be forced to choose a `k` that is actually lower than would be ideal.

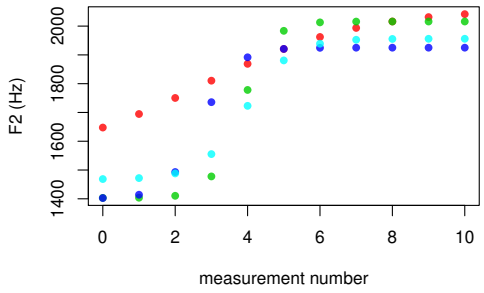
My recommendation is therefore to try and make an informed choice about `k` rather than attempting to rely on a one-size-fits-all strategy (e.g. always using the default value or always specifying a high value for `k`). For instance, formant trajectories for monophthongs and diphthongs tend to be relatively smooth and they don't typically show any meaningful 'high-frequency' wiggliness, so there's no point in setting `k` to a very high value (I'm reluctant to suggest specific values here, but e.g. formant trajectories for single vowels will hardly ever require more than 10 basis functions). An intonation contour for a longer chunk of speech will likely show much more wiggliness, so `k` ought to be set a bit higher too. Setting `k` to a reasonable value requires a bit of experience, so do have a play around with different values of `k` in the examples in the second part of this introduction. `mgcv` also provides some functions and advice for evaluating `k`; see `?gam.check` and `?choose.k` from `mgcv`.

The concepts of basis function and smoothing parameter are closely related to the so-called *estimated degrees of freedom* (or EDF). When the coefficients for basis functions are estimated without any constraints, as in the case of polynomial regression, a smooth with p basis functions uses up exactly p degrees of freedom. However, when the coefficients are constrained by a smoothing parameter, the effective degrees of freedom taken up by the smooth go down (this is because the coefficients for the individual basis functions become dependent on each other). For instance, at extremely high values of the smoothing parameter, the smooth becomes a straight line, which only uses up a single degree of freedom, even if it is represented by more than one basis function. Conversely, at extremely low values of the smoothing parameter, the smooth will use all the potential wiggliness provided by the $\texttt{k} - 1$ basis functions, which corresponds to $\texttt{k} - 1$ degrees of freedom. At intermediate values, the smooth uses an intermediate number of degrees of freedom. The EDF is an estimate of the degrees of freedom that are actually used by a smooth with a given number of basis functions and a given smoothing parameter. Significance tests of smooth terms in GAMs rely on the EDF and not the number of basis functions – and, as a result, model summaries for GAMs always include the EDF. Note that the EDFs for the three models with different `k` values above are very similar (see the plots for the EDF values).

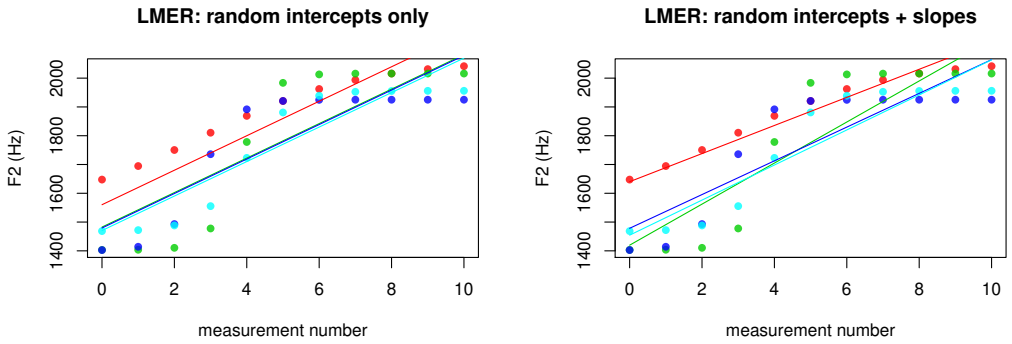
2.2 GAMMs

We are now in a position to move on to GAMMs, that is, generalised additive *mixed* models. This tutorial assumes that the reader is already familiar with random intercepts and slopes from linear mixed effects models and knows how to implement them in R. Just as a reminder: random intercepts in linear mixed models capture by-group random variation in the outcome variable (e.g. between-speaker differences in average F2 values); random slopes capture by-group random variation in the effect of a predictor variable on the outcome variable (e.g. between-speaker differences in the effect of style on F2 – certain speakers exhibit more stylistic adaptation than others). GAMMs are to GAMs as linear mixed effects models are to linear models. That is, GAMMs incorporate random effects alongside parametric and smooth terms. Similar to linear mixed effects models, these random effects can be random intercepts and random slopes. However, GAMMs also offer a third option: random smooths.

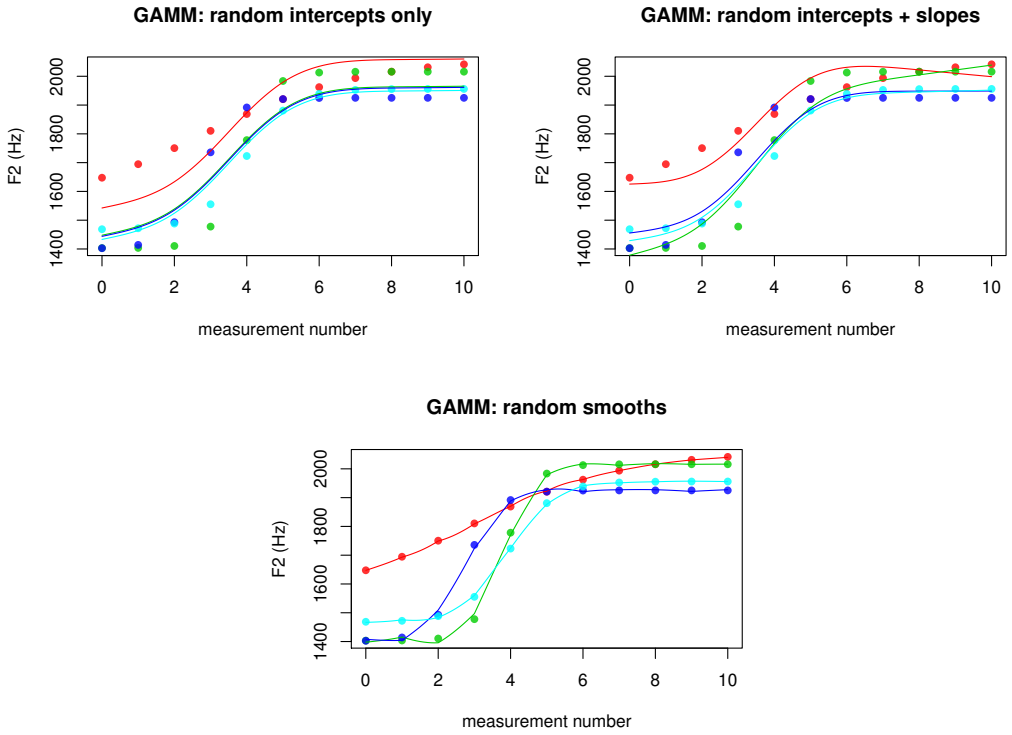
Random smooths are similar to random slopes, but they are more flexible than the latter: while random slopes can only capture by-group variation in linear effects, random smooths can also deal with by-group variation in non-linear effects. An example will help to make this point clearer. Let's assume that the trajectory that we've looked at earlier is an example of a specific vowel, and that we have four further tokens for this vowel. The four trajectories are shown below (this data set is available as `traj_random.csv`):



First, let's use linear mixed effects models to fit straight line approximations to the trajectories. We'll fit two versions of the same model: one with random intercepts only, and a second one with random intercepts and random slopes. The random intercepts model is shown on the left below, while the random intercepts + slopes model is shown on the right.



Unsurprisingly, the model fit is not great. In the random intercepts only model, the fitted lines are parallel to each other, as the slopes are not allowed to vary. In the random intercepts + slopes model, both the height and the slope of the lines vary. Now let's fit three GAMMs to the same trajectories: one with random intercepts only, a second one with random intercepts + slopes and a third one with random smooths.



The model with random intercepts simply varies the height of the lines, but does not yield a particularly good fit. The one with slopes does slightly better: in this case, the same curve is essentially rotated and stretched to match the actual trajectories. Random smooths clearly provide the best fit by fitting individual curves to each trajectory. Note, however, that random smooths are also extremely resource intensive: fitting four separate random smooths to the data requires $4 \times \texttt{k}$ basis functions, and the same number of coefficients need to be estimated.[5]

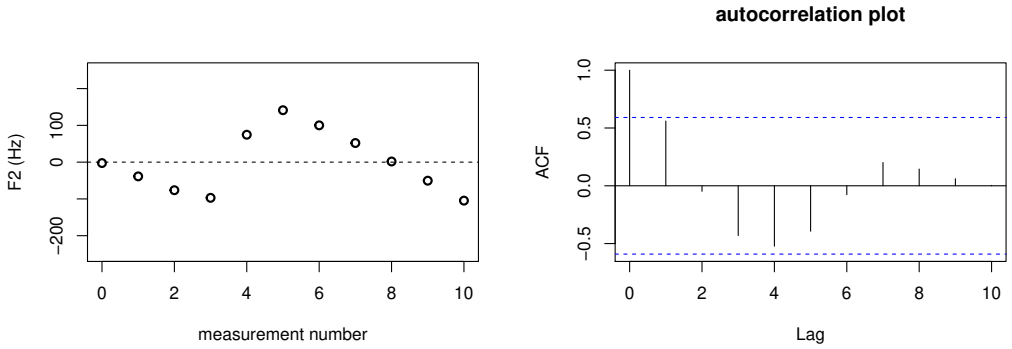
2.3 Residual autocorrelation

Another important concept in GAMM theory is that of *residual autocorrelation*. Consider the example of a linear model fitted to the wiggly trajectory from section 2.1. As explained above, the model fit leaves systematic patterns in the residuals, which are reproduced below for convenience. One way of conceptualising these patterns is through the notion of autocorrelation: the correlation between observed values at fix intervals within a time series, where the size of the interval is usually referred to as the *lag*.[6] For

[5] The separate random smooths each use up `k` rather than $\texttt{k} - 1$ basis functions (as opposed to the smooths that we've looked at before). The additional basis function plays the role of a random intercept. Since there are four smooths (one for each trajectory), the overall number of basis functions is $4 \times \texttt{k}$.

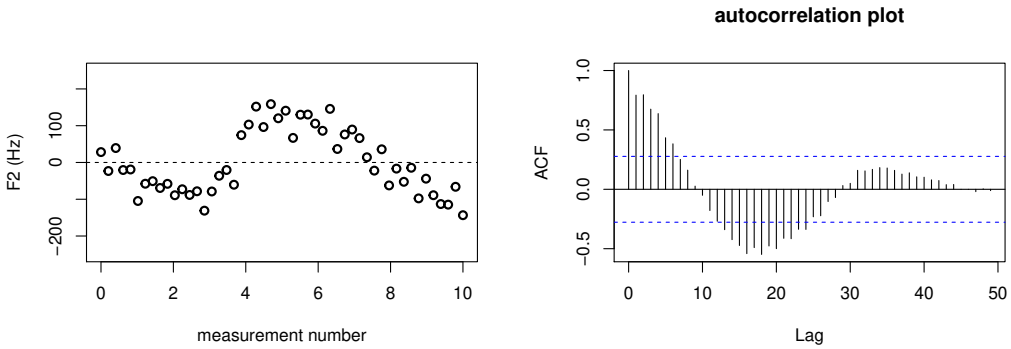
[6] See Venables & Ripley (2002, 390) for the formula for calculating autocorrelation, which is slightly different from the formula for Pearson's correlation coefficient.

instance, the autocorrelation at lag 1 in the residuals below is calculated by taking the f2 values at measurement points 0, 1, 2, ... and correlating them with the f2 values at measurement points 1, 2, 3, ... The autocorrelation at lag 2 is calculated by correlating the f2 values at measurement points 0, 1, 2, ... with those at measurement points 2, 3, 4, ... These autocorrelation values are usually shown in a plot where the horizontal axis shows different lag values and the vertical axis shows the autocorrelation at each lag value. These plots also often include two horizontal lines, which are 'approximate 95% confidence limits' (Venables & Ripley 2002; i.e. autocorrelation values outside these lines can be regarded as significant at $\alpha = 0.05$, though significance isn't of key importance in this case). The figure on the left below shows the residuals from the linear model, while the figure on the right is an autocorrelation plot based on these residuals.



The autocorrelation at lag 0 is simply the correlation of the residuals with themselves, which, of course, always yields a value of 1. The autocorrelation at lag 1 is slightly over 0.5, which is a moderately high value. The formant trajectory is non-linear, but it moves smoothly over time, which means that neighbouring measurements are always relatively close to each other. Since this non-linear movement is not appropriately captured by a linear model, this pattern remains in the residuals, which leads to the observed autocorrelation value. The autocorrelation at lag 2 is very close to 0, but at lags 3, 4 and 5 we observe higher negative values. Negative values indicate a zigzag pattern in the residuals. The fact that negative autocorrelations are only observed at higher lag values indicates that the zigzag pattern occurs not between adjacent measurements but over a slightly longer period. This is indeed what we observe in the residuals: we start with negative values, which become positive at measurement point 4 and then go back to negative at measurement point 9. Note that the zigzag pattern in the current set of residuals is due to the relatively sudden formant transition near the middle of the trajectory, which is entirely smoothed over by the linear model.

Another example of a residual autocorrelation plot is presented below. The residuals come from another linear model, which was fitted to the 50-point trajectory from the previous section (which was introduced in the discussion of different `k` values). The residuals look slightly messier as this trajectory included a small amount of added noise.



Though the overall shape of the autocorrelation plot is similar to that of the previous one, the actual autocorrelation values are not the same. This is especially clear for the first few lag values, where the autocorrelation is really high (close to 0.8). The reason for these high values is that the trajectory moves very slowly when viewed as a function of measurement number – on average, the change between two neighbouring points will be very small. If the trajectory didn't include any noise, the autocorrelation at low lag values would likely be even higher.

Autocorrelation in the residuals is problematic in that it can lead to inaccurate – and often downward biased – standard errors, confidence intervals, and *p*-values. There are two ways of dealing with this issue (cf. Baayen et al. 2016): (i) fitting more accurate models that capture all patterns in the residuals (often through using random smooths) or (ii) including a so-called *error model* in the regression, which essentially adjusts the model output to control for the biasing effect of residual autocorrelation. Both of these methods will be discussed in more detail below.

2.4 Why do we need random smooths and/or error models?

Why are patterns in the residuals such a serious issue for regression models? And why do we need to add random effects and/or error models to our GAM(M)s? These questions are partly answered in standard texts dealing with mixed effects regression models (Baayen, 2008; Gelman & Hill, 2007; Zuur et al., 2009). More detailed treatments are given in Barr et al. (2013) and Bates et al. (2015), while Winter (2013) provides a particularly clear explanation that should be fairly easy to follow for readers with no background in statistics. The focus in many of these texts is on how models without random effects may violate the underlying assumptions of regression models, in particular the assumptions of *linearity* and *independence of errors*. What I'd like to do here is offer a slightly different approach to the question, and discuss the importance of random effects in more intuitive terms.

Regression models attempt to make educated guesses about 'hidden' underlying parameters (which correspond to properties of interest in the real world) based on observable data. Each of the data points in a data set provides some information about these parameters, increasing our confidence in the guesses the model makes about them (we will refer to these guesses as estimates). However, it turns out that individual data points don't always provide the same amount of information about the underlying parameters. In other words, two data sets with the same number of observations may provide different amounts of information about the same parameters. These differences are

often dependent on the presence or absence of grouping structure and temporal/spatial structure in the data.

In terms of the amount of information per data point, the best case scenario is a data set where there is no grouping or temporal/spatial structure (beyond the variables whose effects we want to estimate). In such data sets, a regression model can assume that individual data points are determined solely by the underlying parameters plus a bit of random noise, so all the information in a given data point can be used towards estimating the underlying parameters. Let's illustrate this point using the formant trajectories from section 2.2. Our data set would have this structure (no grouping/temporal/spatial structure) if each of the measurements (i.e. each of the dots in the graphs above) came from a separate vowel. This would mean that we would have to sample 44 different vowel tokens, and only take a single measurement at a random time point for each of them (this would be admittedly a rather weird data set). If we were to, say, double the number of sampled vowels, our confidence in our estimates would also increase – each new data point would contribute additional information about the underlying parameters.

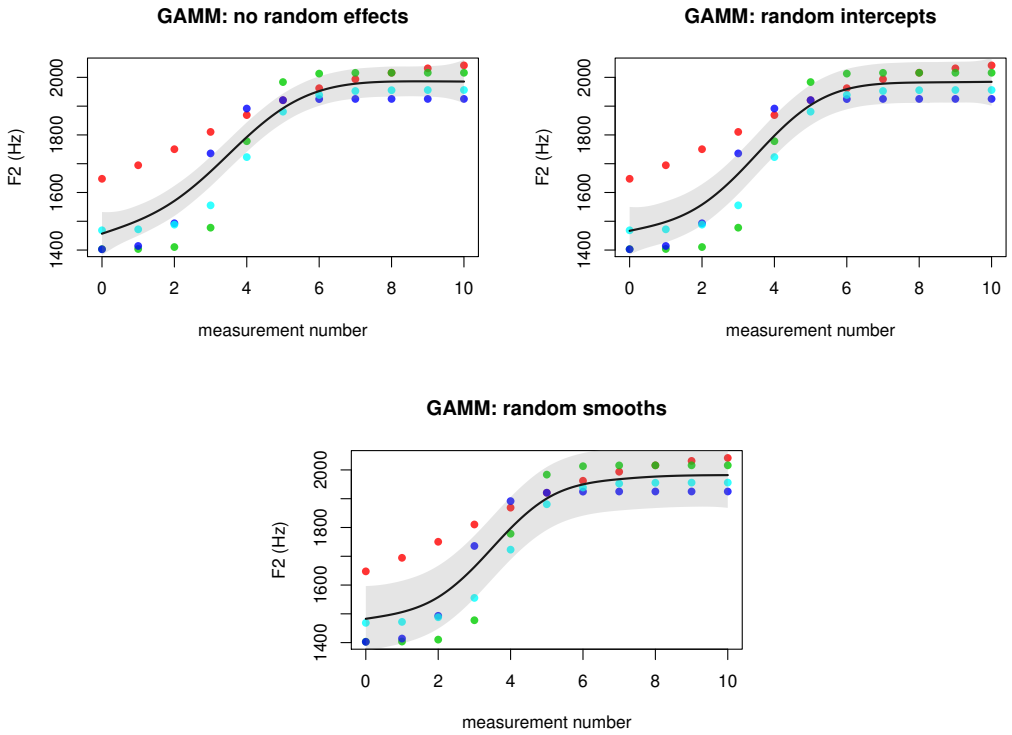
However, that is not the way our data set is structured. The 44 measurements actually come from only 4 vowels, and measurements taken from a single vowel are not independent of each other. Each of the data points is determined by a combination of factors: the underlying parameters, which trajectory it is in and what values the neighbouring points take on (the last point reflects the fact that there is almost always some autocorrelation in time-series data). Therefore, only part of the information in a given data point can be used towards estimating the underlying parameters – the rest of the information is actually about other stuff that we might not be interested in. An increase in the size of this data set wouldn't necessarily increase our confidence in our estimates of the underlying parameters. For instance, if we doubled the size of the data set by taking another 11 measurements along the *same* 4 trajectories at different time points, we wouldn't gain much additional information about the parameters that define the underlying curve – instead, we would gain more information about the individual trajectories themselves. If, on the other hand, we added measurements from 4 additional vowels, that would contribute more information towards the underlying parameters.

When we fit a regression model to our data set without random effects or an appropriate error model, we essentially ignore the grouping/temporal/spatial structure in the data: we pretend that the data set consists of independent measurements, each of them taken from a separate vowel. As a result, we will also erroneously assume that all the information in the individual data points can be used towards estimating the underlying parameters, even though some of the information is actually about other things like the separate trajectories. Since we think that we have more information that we actually do, we become overconfident about our estimates, which leads to anti-conservative results: p-values that are biased downwards and overly narrow confidence intervals. When the random effects / error model are correctly specified, the model uses the right amount of information towards estimating the underlying parameters, and the overconfidence issue disappears.

We run into the same problem when we use random intercepts and slopes to fit straight lines to non-linear trajectories. The straight lines correctly 'soak up' some of the trajectory-specific information, but not all of it: they can't deal with non-linear dependencies, so those remain in the data. As a result, some of the information that

we use towards estimating the underlying parameters will still be about the individual trajectories, so our model remains overconfident.

The three graphs below illustrate this point by plotting 95% confidence intervals for three models: one without random effects, one with random intercepts only and one with random smooths.



The confidence interval becomes slightly wider when we move from the no random effects model to the random intercepts one, and substantially wider for the random smooths model. Although the confidence interval for the final model may seem too wide, it is probably more accurate than the other two. After all, we need to bear in mind that this estimate is really only based on 4 vowels. For comparison, how confident would you be about a group mean based on 4 measurements?

2.5 Different smooth classes

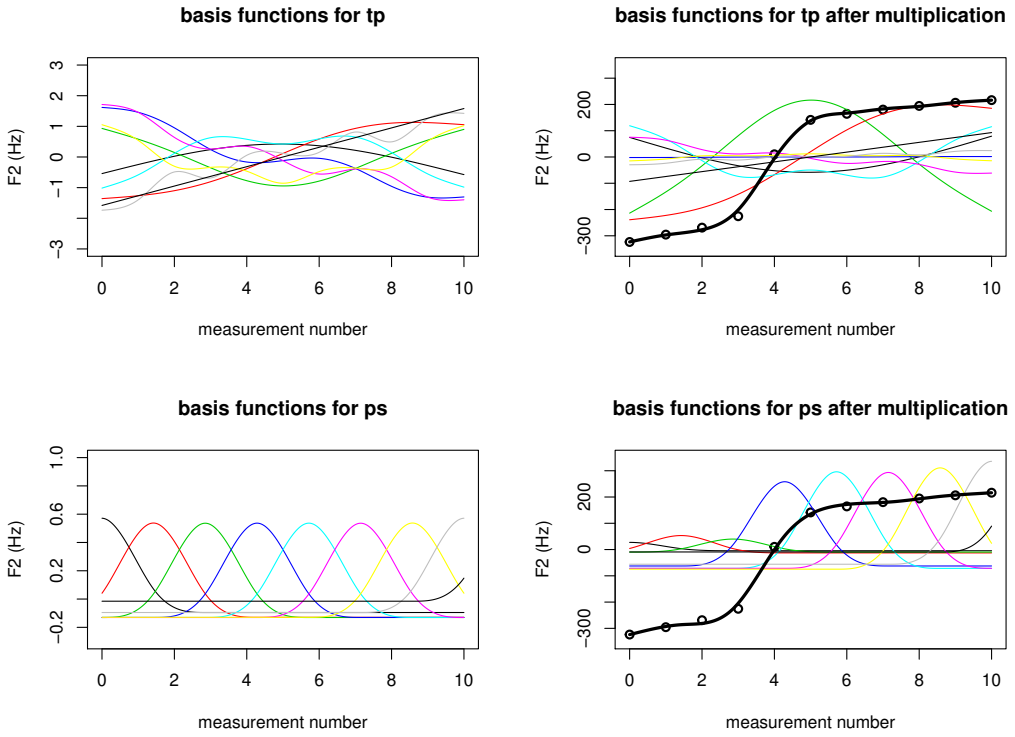
Now that we know what GAMMs are, how they are different from GAMs and why random smooths / error models are necessary, it is time to introduce the concept of different smooth classes. We've actually already seen a range of different smooth classes in action, but only one of these has been mentioned explicitly: cubic regression splines. In what follows, we'll look at what distinguishes smooth classes from each other and consider a few examples.

Smooth classes are mainly defined by the basis functions used to generate the smoothers (and also by the type of *smoothing penalty* applied to them, but we won't discuss this). The graphs on page 5 show the basis functions for a smooth that belongs to the class of cubic regression splines. The models below exemplify two further smooth classes: thin plate regression splines (`bs="tp"`) and P-splines (`bs="ps"`; P-splines is

short for 'penalised B-splines' or 'penalised basis splines').

```
demo.gam.tp <- bam(f2 ~ s(measurement.no, bs = "tp"), data = traj)
demo.gam.ps <- bam(f2 ~ s(measurement.no, bs = "ps"), data = traj)
```

While the exact differences among smooth classes are not so important for us, it is instructive to compare their basis functions and the fitted curves. The figures below show the basis functions for thin plate regression splines and P-splines before and after multiplication by the model coefficients, and should be compared with the cubic regression splines on page 5.



The basis functions can be very different (compare e.g. `tp` vs. `ps`), but the overall smooths are quite similar across the three models. P-splines seem to provide a slightly (though only very slightly!) worse fit in this case, while cubic and thin plate regression splines do equally well. The default smooth for the `bam()` function from the `mgcv` package is the thin plate regression spline.

The smooths that we've looked at so far are all univariate smooths: they fit a smooth line to the outcome variable as a function of a single predictor. However, it is possible to specify multivariate smooths as well, and certain smooth classes are capable of representing such smooths. For instance, one might want to look at how vowel duration affects the shape of F2 trajectories for a given vowel. One way to do this is to specify a bivariate smooth, where one of the predictor variables is `measurement.no` (where the measurement was taken along the trajectory), and the other one `duration` (the overall duration of the vowel in seconds). The fitted bivariate smooth will be a two-dimensional surface, which essentially consists of trajectory shapes that vary smoothly as a function of overall duration. We'll see examples of multivariate smooths in section 3.

One final note about smooth classes. GAMMs can use smooths to represent not only fitted curves and surfaces, but also random intercepts and slopes. For instance, the linear mixed model with intercepts and slopes on page 9 can be specified in two different ways: using the traditional mixed model specification from the `lme4` package (Bates et al., 2011), or using a GAMM model specification.[7]

```
demo.slope.lmer <- lmer(f2 ~ measurement.no +
                            (1 + measurement.no || vowel),
                        data=traj.random)

demo.slope.gamm <- bam(f2 ~ measurement.no +
                           s(vowel, bs="re") +
                           s(vowel, measurement.no, bs="re"),
                       data=traj.random)
```

These formulae actually specify the same model. Random intercepts and slopes are represented by the random effects (`bs="re"`) smooth class in GAMMs, and they behave in much the same way as random effects in linear mixed effects models. Random smooths are a different type of construct, and they do not have a straightforward equivalent in linear models. The smooth class for random smooths is called *factor smooth interactions* (`bs="fs"`). Its use will be illustrated later in section 3.

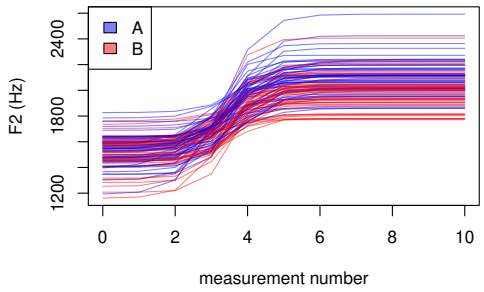
2.6 Significance testing using GAMMs

2.6.1 Methods for significance testing with GAMMs

The models and data discussed in the previous section are useful in that they illustrate some of the basic concepts of GAMMs, but they are also a bit weird. There is only a single group of trajectories, and the shape of these trajectories does not vary as a function of any other predictors (although the individual F2 measurements do, of course, vary as a function of time). This type of situation does not typically arise in real examples: dynamic analyses of linguistic data are usually conducted with the goal of testing whether a given set of predictors has a significant effect on the trajectories under investigation. For instance, one might look at whether the shape of F2 trajectories is affected by vowel duration, whether pitch contours are different across questions and statements, or whether a diachronic change in spectral centre of gravity for a given fricative follows different patterns in different communities. While significance testing is relatively straightforward for linear models (though less so for linear mixed models), GAMMs offer a number of different ways of testing for significance. In this section, we briefly review these methods. It should be emphasised that they are not all equally appropriate, and some of them are, in fact, seriously flawed. The next section identifies potential issues with these methods and outlines a few recommendations about how they should be used.

[7] The random effect specification in the LMER model is `(1 + measurement.no || vowel)` instead of the more typical `(1 + measurement.no | vowel)`, which results in a model where the variance components for the random intercept and slope are estimated, but their correlation isn't. This is because linear mixed models specified using `bam()` do not include correlations between random slopes and intercepts under the same grouping factor.

We start with a simple scenario. Let's say we have two words sharing the same diphthong, and we suspect that the realisation of the diphthong differs between the two words. We'll refer to the words as *A* and *B*. We collect dynamic F2 measurements for 50 tokens of each word. The two sets of 50 trajectories are shown below:



We use a GAMM of the following structure to test for significant differences between the two words:

```
demo.w.gamm <- bam(f2 ~ word +
                         s(measurement.no) +
                         s(measurement.no, by=word) +
                         s(measurement.no, traj, bs="fs", m=1),
                   data=dat.words, method="ML")
```

Let's go through this model specification quickly. The first predictor is a parametric term that captures overall differences in the height of the trajectories as a function of the word they come from. The second predictor, `s(measurement.no)`, corresponds to a single smooth fit at the reference value of the categorical predictor `word` (i.e. `A`). The third predictor is a so-called *difference smooth* that captures the difference between the trajectories for `A` and `B`. We will discuss difference smooths in more detail later. The last predictor corresponds to random smooths by trajectory (i.e. a separate smooth for each of the 50 trajectories). Again, we'll say more about these later.

Confusingly, there are at least *six* different ways of testing whether the difference between *A* and *B* is significant (and probably more). Four of these are available as part of standard model summaries, and the other two can be performed by plotting confidence intervals.

Let's start with the model summary-based methods. Here is the model summary for `demo.w.gamm`:

```
summary(demo.w.gamm)

##
## Family: gaussian
## Link function: identity
##
## Formula:
## f2 ~ word + s(measurement.no) + s(measurement.no, by = word) +
##     s(measurement.no, traj, bs = "fs", m = 1)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1898.88      12.22 155.355  < 2e-16 ***
## wordB         -80.89      17.12  -4.725 3.58e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                              edf  Ref.df       F p-value
## s(measurement.no)          8.936   8.953 275.314 < 2e-16 ***
## s(measurement.no):wordB    3.378   3.618   4.566 0.00272 **
## s(measurement.no,traj)   793.816 898.000  81.655 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.997   Deviance explained = 99.9%
## -ML = 5908.6  Scale est. = 240.07    n = 1100
```

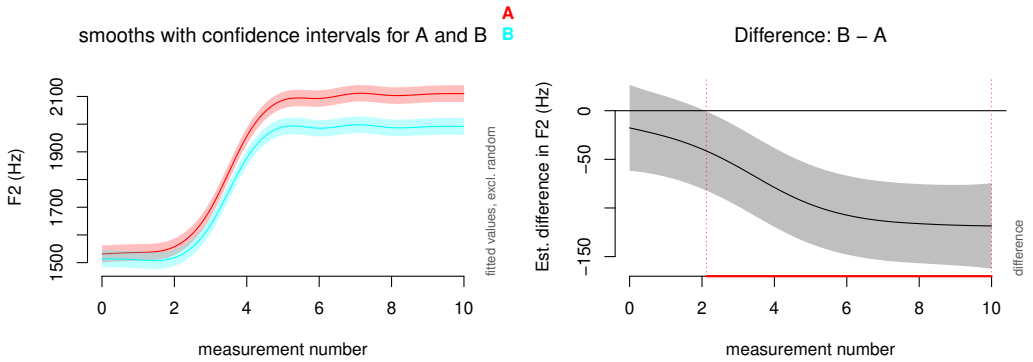
The initial section up to the first table should be familiar from other types of regression models. The `Parametric coefficients` table shows all non-smooth terms, that is, the intercept and the categorical predictor `word`. The next table (`Approximate significance of smooth terms`) summarises the smooth terms: the reference smooth, the difference smooth and the random smooths. Both the parametric and the smooth tables show p-values for the terms, although they are based on different tests: t-tests for parametric terms and 'approximate' F-tests for smooth terms. Wood (2006, 191) warns that the approximate p-values values for smooth terms should be taken with a pinch of salt: they can be anticonservative in certain cases.

So which p-values should we be looking at? First, we can look at the p-value for the parametric term `word` (method 1). Assuming an alpha-level of 0.05, this p-value is $< \alpha$, which means that there is a significant difference in the overall height of the two trajectories. We can also look at the p-value of the difference smooth, which, again, is < 0.05 – that is, there is a significant difference between the shapes of the two trajectories (method 2). Another option is to claim a significant difference between the two trajectories if either the parametric term or the difference smooth (or both) are significant (method 3). The last non-visual option is to set up a nested model which excludes the parametric term and the difference smooth, and compare this to the original model using the `compareML()` command from the `itsadug` package, which is actually a form of `anova` (method 4):

```
demo.w.gamm2 <- bam(f2 ~ s(measurement.no) +
                        s(measurement.no, traj, bs="fs", m=1),
                    data=dat.words, method="ML")
compareML(demo.w.gamm, demo.w.gamm2, print.output=F)$table

##          Model    Score Edf  Chisq    Df   p.value Sig.
## 1 demo.w.gamm2 5923.394   5
## 2  demo.w.gamm 5908.614   8 14.780 3.000 1.707e-06  ***
```

According to the model comparison, the inclusion of the parametric term and the smooth difference term significantly improves the model fit. Importantly, these methods tell us little about the exact nature of the difference between the two words.

There are two visual methods for significance testing, both of which rely on confidence intervals. First, we can plot the predicted trajectories for both words with corresponding pointwise confidence intervals and check for overlap / lack of overlap at different points (method 5). Second, we can plot the difference smooth itself along with a confidence interval and check whether the confidence interval includes 0 at different points (method 6). One of the advantages of these methods is that they allow us to see where and in what way the trajectories words differ. These methods are illustrated below:



Both methods suggest that there is a significant difference between the two sets of trajectories. Moreover, they also reveal that the trajectories only diverge after measurement number 2 (that's the 3rd measurement out of 11). In other words, the trajectories differ significantly, but only between measurement numbers 2 and 10.

As a final note, significance testing with GAMMs can be substantially more complicated when the predictor of interest is a continuous variable. Although the general principles discussed in this section apply to continuous variables as well, their implementation can be a lot trickier due to the potential complexity of smooth interactions and constraints on the software packages used to fit GAMMs. We will go through a few worked examples later in the tutorial.

2.6.2 Recommendations

In Sóskuthy (2016) and Sóskuthy (in prep), I present simulation-based results that reveal a wide range of variation in the rate of false positives and false negatives across the different methods discussed above. A point-by-point summary of these results is presented below:

- Significance testing based on the t/F-values for the parametric / smooth terms in the model summary is only justified when our predictions are directly about these individual terms. For example, if our prediction is that the average value of a trajectory will be higher in one condition than in another, but we have no predictions about trajectory shapes, we can safely rely on the p-value for the parametric term (method 1). Conversely, if our prediction relates to the shapes of the trajectories but does not concern their average value, we can use the p-value for the smooth term (method 2).

- Although predictions specifically about the parametric / smooth terms do arise occasionally, a lot of the time researchers are interested in overall differences between trajectories regardless of whether those differences are in their average value or their shape. In such cases, it is tempting to declare significance if either the parametric term or the smooth term is significant (method 3). However, this leads to higher-than-nominal false positive rates (just below 0.10 at $\alpha = 0.05$). Model comparison where both the parametric *and* the smooth terms are excluded in the nested model (method 4) yields close-to-nominal false positive rates. Another way to avoid this issue is through correction for multiple comparisons (two comparisons in this case), though this can lead to substantially diminished power. This introduction relies mainly on method 4, and correction for multiple comparisons won't be discussed in detail.

- Both visual methods (5 and 6) suffer from an anti-conservativity issue. If we report significant differences whenever there is *any* point where confidence intervals are non-overlapping (method 5) / the confidence interval for the estimated difference excludes 0 (method 6), the rate of false positives is too high (around 0.12 in the simulations at $\alpha = 0.05$). The rate of false positives decreases if we require significant differences at more than one point to report a significant overall difference, but this requires an arbitrary decision about the number of points, and can easily lead to results that are too conservative.

- Comparison based on whether there is overlap between two confidence intervals (method 5) is inadequate for significance testing, and graphs with separate trajectories (rather than a difference smooth) should only be used for graphical illustration. People tend to misinterpret the meaning of overlapping confidence intervals in such graphs. When the confidence intervals do not overlap, there is indeed a significant difference between the estimated quantities. However, we cannot make any conclusions about the significance of the difference when the confidence intervals do overlap: it may be significant but it may also be non-significant. Difference smooths (method 6) do not suffer from this problem, and should be the preferred option. Note that this issue is independent of the anti-conservativity problem described above.

- The simulations also reveal that failing to include (i) an error model to adjust the model output for residual autocorrelation within the trajectories or (ii) random smooths-per-trajectory can lead to catastrophically high false positive rates (up to 0.6 at $\alpha = 0.05$) regardless of what method is used for significance testing. These methods are discussed in more detail below.

Based on these results, the most reliable (i.e. least anti-conservative) option for significance testing is to first use an ANOVA (method 4) to see if there is an overall difference between groups of trajectories, and then look at difference smooths (method 6) to identify where the difference lies along the trajectory. Additionally, it is also useful to plot individual smooths (with or without confidence intervals) to provide a visual summary of the main trends in the data set, but these should not be used for significance testing.

3 A GAMM tutorial

In this tutorial, we will work through two detailed examples. The first of these is based on simulated data, while the second one uses a real data set taken from Stuart-Smith et al. (2015).

3.1 Analysing a simple simulated data set

The first data set contains simulated F2 trajectories, and is very similar to the example data set introduced in section 2.6.1. It contains 50 F2 trajectories, each of them represented by 11 measurements taken at equal intervals (at 0%, 10%, 20%, ..., 100%). The observations in the data set are the individual measurements, which means that there are 550 data points altogether. The variable `measurement.no` codes the location of individual data points along the trajectory. Each of the trajectories has an ID (a number between 1–50), which is encoded in the column `traj`. The trajectories represent two different words: 25 of them come from word *A* and 25 of them from word *B*. This is encoded by the `word` variable. The underlying curves that served as the basis of the simulated trajectories overlap at the beginning, but are different by about 100 Hz near the end. There is an additional variable termed `duration`, which stands for overall vowel duration measured in seconds. The simulation was set up so that long vowels have slightly wider trajectories than short vowels. The data set is called `words.50` and can be downloaded from the github page for this introduction:

https://github.com/soskuthy/gamm_intro.

Here's a small sample of the data:

```
head(words.50)
```

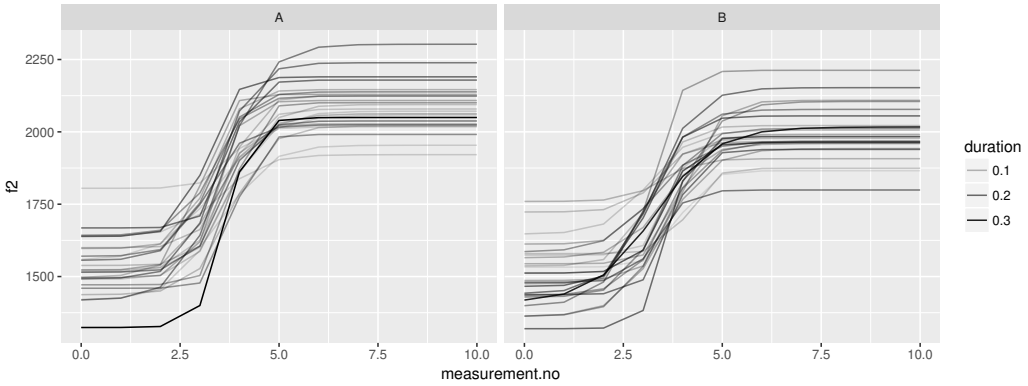
```
##     traj word measurement.no       f2  duration
## 1 traj.1    A              0 1642.761 0.1378182
## 2 traj.1    A              1 1644.162 0.1378182
## 3 traj.1    A              2 1659.948 0.1378182
## 4 traj.1    A              3 1788.793 0.1378182
## 5 traj.1    A              4 2044.977 0.1378182
## 6 traj.1    A              5 2115.984 0.1378182
```

Let's import the libraries that we'll be using.

```
library(ggplot2)
library(mgcv)
library(itsadug)
source("gamm_hacks.r")
```

First of all, let's create a simple plot to see what the raw data look like. This is good practice regardless of the type of model one is planning to fit, but it is especially useful for GAMMs, where the shape of the trajectories has implications for the model fitting procedure (e.g. choosing the number of basis functions). We'll use the package `ggplot2` to create the plot, as it makes it really easy to show the structure of the data set through the use of colours and other devices. The trajectories representing the two words are shown in separate panels and the overall duration of each trajectory is indicated by shading: longer trajectories are darker (the trajectories are time-normalised, so they all have the same length along the x-axis). Since this tutorial is not about `ggplot2`, we won't discuss the code below.

```
ggplot(words.50, aes(x=measurement.no, y=f2, group=traj,
                     alpha=duration)) +
  facet_grid(~word) + geom_line()
```



So what do we want to capture in our model? First, we want to fit separate smooths to the two trajectories, and we'll want to use model comparison and difference smooths to see whether they are different. We also want to include some type of interaction between duration and the shape of the trajectories. Finally, we want to include (i) random smooths by trajectory and (ii) a residual error model in order to avoid false positives and obtain more accurate estimates of the underlying curves.

Let's start with a very simple model that fits separate smooths to the two words. There are a number of different ways of doing this; we'll look at two of these. First, we can specify a model that simply includes two smooths: one for word *A* and another one for word *B*. Though this is probably the simplest way of fitting this model, it's not necessarily the most useful one. The second method is to fit one smooth to word *A* and then another smooth that represents the *difference* between *A* and *B*. This is identical to the first model in terms of the model fit, but it is easier to interpret the model output when a difference smooth is included, as it tells us directly whether there is a significant difference between the shapes of the two trajectories. Both types of models are shown below along with relevant bits of the model summary. The models use cubic regression splines (`bs="cr"`) with the default number of knots (`k=10`). Using different basis functions does not substantially alter the results. The models are fitted using maximum likelihood estimation (`method="ML"`), which is necessary since we want to perform model comparison between models with different fixed effects. The default method for `bam()` is fREML or fast restricted maximum likelihood estimation, but this cannot be used for comparing models with different fixed effects.

```
# model with separate smooths

words.50$word <- as.factor(words.50$word)
words.50.gam.sep <- bam(f2 ~ word + s(measurement.no, by=word, bs="cr"),
                        data=words.50, method="ML")
summary.coefs(words.50.gam.sep)

## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1885.149      5.755 327.588  < 2e-16 ***
## wordB        -67.372      8.138  -8.278 1.01e-15 ***
##
## Approximate significance of smooth terms:
##                            edf Ref.df     F p-value
## s(measurement.no):wordA 7.311  8.258 213.0  <2e-16 ***
## s(measurement.no):wordB 6.832  7.871 169.4  <2e-16 ***

# model with smooth for A & difference smooth

words.50$word.ord <- as.ordered(words.50$word)
contrasts(words.50$word.ord) <- "contr.treatment"
words.50.gam.diff <- bam(f2 ~ word.ord + s(measurement.no, bs="cr") +
                           s(measurement.no, by=word.ord, bs="cr"),
                         data=words.50, method="ML")
summary.coefs(words.50.gam.diff)

## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1885.149      5.735 328.713  < 2e-16 ***
## word.ordB    -67.372      8.110  -8.307 7.99e-16 ***
##
## Approximate significance of smooth terms:
##                                edf Ref.df      F p-value
## s(measurement.no)            7.805  8.594 234.40 < 2e-16 ***
## s(measurement.no):word.ordB 1.040  1.079  10.84 0.00098 ***
```

`summary.coefs()` (a function from `gamm_hacks.r`) is used to save space by excluding parts of the model summary. Readers are encouraged to use the standard `summary()` function with GAMMs, as it includes some additional useful information about the model fit.

Fitting the model with two separate smooths is easy: we need to include a parametric term that captures overall differences between the trajectories,[8] and add the option `by=word` to the smooth term (which asks for separate smooths to be fit at each level of the factor `word`).

Models with difference smooths require a different approach. First, the categorical grouping variable for the words needs to be converted to a so-called 'ordered factor' using `as.ordered()`, and the contrasts for this ordered factor need to be set to `"contr.treatment"` (this latter step is important, or otherwise the model estimates will be off). Second, the model formula needs to include (i) a parametric term for `word.ord`, (ii) a smooth over `measurement.no` without any grouping specification and

[8] This parametric term needs to be included in both models. Otherwise, the models could not capture overall differences in F2 and would (incorrectly) force both fitted smooths to have the same average value over the trajectory.

(iii) a smooth over `measurement.no` with the grouping specification `by=word.ord`.

In the first model summary, the separate smooths simply represent the two different words. The significance values in the model summary refer to the individual terms: they suggest that both curves are significantly different from 0 (i.e. not simply flat lines). However, they do not tell us anything about the difference between the two terms. They could both be significant even if the two underlying curves were exactly the same. In the second model summary, the term `s(measurement.no)` represents the *reference smooth*, that is, a curve fit to trajectories at the reference level of the ordered factor `word.ord` (i.e. *A*). The term `s(measurement.no):word.ordB` represents the difference smooth, that is, the difference between the trajectories for *A* and *B*.

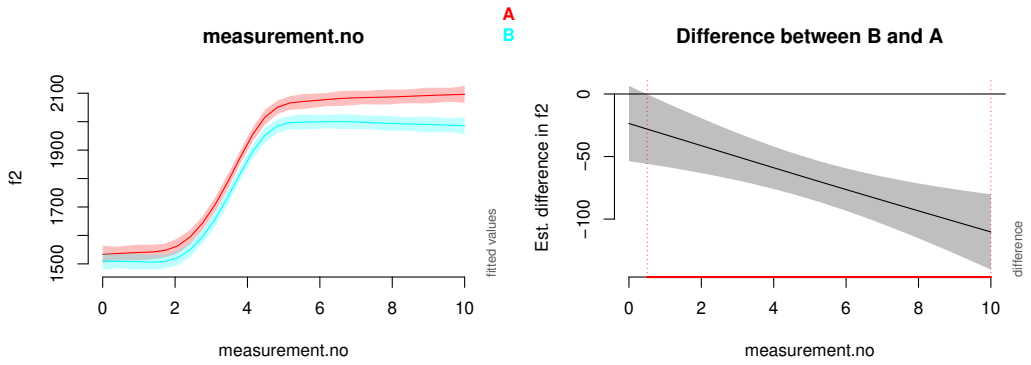
The fact that the difference smooth and the parametric term for `word.ord` are both significant suggests that the trajectories for the two words are indeed different. In order to confirm this, we can perform model comparison using the function `compareML()`. Following the recommendations in section 2.6.2, the nested model excludes both the parametric term and the smooth difference term (i.e. all terms that relate to the effect of `word.ord`).

```
# fitting a nested model without the difference smooth
words.50.gam.diff.0 <- bam(f2 ~ s(measurement.no, bs="cr"),
                           data=words.50, method="ML")

# model comparison using the compareML() function from itsadug
# this is very similar to the anova() function, but better suited
# to models fitted using bam(); some parts of the output are suppressed
compareML(words.50.gam.diff, words.50.gam.diff.0, print.output=F)$table

##                  Model    Score Edf  Chisq    Df   p.value Sig.
## 1 words.50.gam.diff.0 3334.300   3
## 2   words.50.gam.diff 3296.538   6 37.762 3.000 2.798e-16  ***
```

The model comparison suggests that the inclusion of the difference smooth improves the model fit significantly.

Let's create a plot of the model predictions and the difference smooth. These can be generated using the `plot_smooth` and the `plot_diff` functions from `itsadug`.

```
plot_smooth(words.50.gam.diff, view="measurement.no",
            plot_all="word.ord", rug=F)
plot_diff(words.50.gam.diff, view="measurement.no",
          comp=list(word.ord=c("B","A")))
```



The `view` option determines what variable to show along the x-axis; the `plot_all = "word.ord"` option tells R to plot predictions separately for each value of the factor `word.ord`; the `rug=F` option tells R not to include ticks for each data point along the x-axis (these can make plotting very slow and plot files very large when there are many thousands of data points); and the `comp` option specifies the levels of `word.ord` that the difference smooth is based on. In this case, the difference smooth shows $B - A$. Note that the confidence interval for the difference smooth seems a bit too narrow: the underlying curves were specified in such a way that there is no actual difference between the curves until about `measurement.no` 2, but the difference smooth shows a significant difference along almost the entire trajectory.

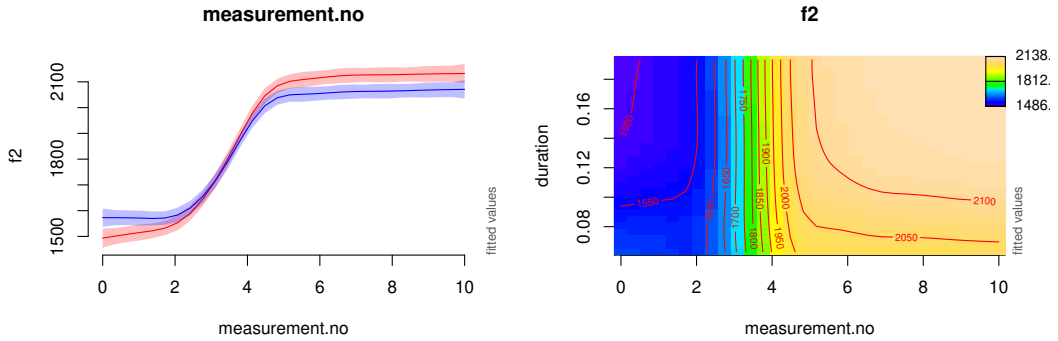
As a second step, let's try to account for the influence of `duration` on the trajectories. Simply including it in the model as a parametric term or even as a smooth on its own won't work: its effect is not on average F2 values, but on the shapes of the trajectories. So what we really want is a non-linear interaction between `duration` and the smooths for `measurement.no`. Moreover, it would be useful if we could separate this interaction term from the main effects of `duration` and `measurement.no`. The solution is to use so-called 'tensor product interactions'. Although the name is quite intimidating, these are conceptually very similar to interactions in linear models. All we need to do is include three terms: `s(measurement.no)` (a smooth for the main effect of `measurement.no`), `s(duration)` (main effect of `duration`) and `ti(measurement.no, duration)` (the interaction between the two variables).

```
words.50.gam.dur <- bam(f2 ~ word.ord + s(measurement.no, bs="cr") +
                            s(duration, bs="cr") +
                            ti(measurement.no, duration) +
                            s(measurement.no, by=word.ord, bs="cr"),
                        data=words.50, method="ML")
summary.coefs(words.50.gam.dur)

## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1887.721      5.315 355.170   <2e-16 ***
## word.ordB    -72.516      7.595  -9.548   <2e-16 ***
##
## Approximate significance of smooth terms:
##                                edf Ref.df       F  p-value
## s(measurement.no)            7.958  8.681 267.519  < 2e-16 ***
## s(duration)                  3.339  4.014   7.821 3.78e-06 ***
## ti(measurement.no,duration) 6.409  8.551   8.222 6.08e-11 ***
## s(measurement.no):word.ordB 1.662  2.062  10.137 3.91e-05 ***
```

The interaction term is significant according to the model summary. If we wanted to run a more rigorous test, we could fit a nested model where both smooths that include `duration` are dropped, and then compare it to the full model using `compareML()`.

There are two ways to plot this interaction. First, we can plot smooths at a few different values of `duration`. Or, alternatively, we can plot the surface that represents the interaction between `duration` and `measurement.no` using colours (with warmer colours representing higher values). Both options are shown below:

```
plot_smooth(words.50.gam.dur, view="measurement.no", cond=list(duration=0.16),
            rug=F, col="red")
plot_smooth(words.50.gam.dur, view="measurement.no", cond=list(duration=0.08),
            rug=F, col="blue", add=T)
fvisgam(words.50.gam.dur, view=c("measurement.no","duration"),
        ylim=quantile(words.50$duration, c(0.1, 0.9)))
```



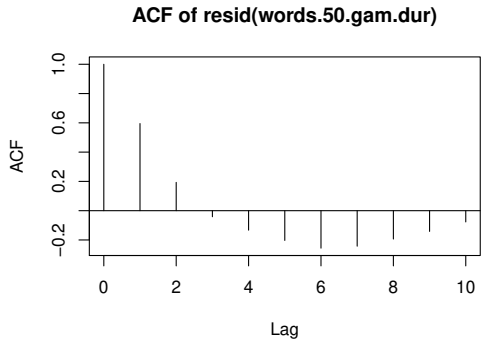
The function `plot_smooth` needs to be run twice to generate the separate smooths: once to create the plotting area and a smooth for longer trajectories (with a duration of 0.16 s, set using the `cond` option; this smooth is shown in red) and once to add another smooth for shorter trajectories (with a duration of 0.08 s; shown in blue). Adding a smooth to an existing plot is done by including the `add=T` option. The result is the plot on the left, which suggests that shorter vowels have a flatter f2 trajectory.

The plot on the right is not easy to interpret, but let's give it a go. First, let's pick a specific point along the *y*-axis, say, 0.1 s (the second notch from the bottom). This will allow us to look at a predicted trajectory for a given duration value. Now imagine a horizontal line crossing this point, and inspect the colours along this line: we start with dark blue and move towards warmer colours, finally arriving at light ochre. This indicates a rise in values, which is also shown by the labelled contour lines (which should be read similarly to contour lines in topographic maps). The colour doesn't change much between measurement points 0–2 and between 5–10, indicating that the predicted trajectory is stable in these ranges, and that most of the change takes place between 2–5. Now let's zoom out a bit and try to look at a range of different duration values at the same time. What we see is that the trajectory starts relatively high (over 1550 Hz) at short durations, and a bit lower (close to 1500 Hz) at long durations. Conversely, the trajectory ends high for long durations (over 2100 Hz), but ends low for short durations (lower than 2050 Hz). In the current case, this heatmap is not particularly useful, as the difference between trajectories with different durations is relatively small. However, it is often worth inspecting both types of graphs to get a better sense of what the model fit actually looks like.

Though our model already includes a lot of nuance, it is not yet complete: in its current form it does not recognise the fact that the data consists of a set of 50 trajectories rather than 550 completely independent measurements, and is therefore likely overconfident in its estimates. In other words, the model is not capturing dependencies within individual trajectories, which likely leave patterns in the residuals. This should show up in a residual autocorrelation plot, so let's create one. So far, we have only looked at autocorrelation within a single residual series. In the current case, there are

50 separate trajectories with 50 separate residual series (though the model is not aware that the residuals come from different trajectories), so it is impractical to create separate autocorrelation plots for each of them. Instead, we will look at average autocorrelation values across the 50 residual series. The plot is shown below.

```
acf_plot(resid(words.50.gam.dur), split_by=list(words.50$traj))
```



The function `resid(words.50.gam.dur)` extracts the residuals from the model. These are the main argument to the function `acf_plot()`, which creates an averaged residual plot for multiple trajectories. Since the residuals are just a series of numbers without any structure, the function needs to be told how to chop them up into separate trajectories: this is done using the option `split_by=list(words.50$traj)`, which tells `acf_plot()` to calculate the autocorrelations separately for each trajectory and then average over them. Note also that `acf_plot()` (like most other autocorrelation plotting functions) assumes that the order of the observations in your data frame (`words.50`) is the same as their order in the time series over which the autocorrelations are calculated (so e.g. an observation at `measurement.no = 2` comes before an observation from the same trajectory at `measurement.no = 5`).

The residual autocorrelation plot suggests that there is a fair bit of positive autocorrelation at lag 1. This confirms our suspicion that there are patterns in the residuals of the model, which are likely affecting the model output as well. The remaining autocorrelation values are somewhat less worrying with low values mainly between 0.2 and −0.2.

We'll look at two solutions for addressing this issue: (i) using random structures and (ii) using an autoregressive error model within trajectories. Both of these are effective at keeping type I error rates close to nominal values in type I error simulations using data sets with a structure very similar to the current one (Sóskuthy, in prep).

Let's start with the random structures: we'll try out a few different options and perform model comparisons to see what type of random structure is best suited to our data. We'll explore three different options: (i) random intercepts only, (ii) random intercepts plus slopes and (iii) random smooths. The code for fitting these models is shown below. The last model may take a while to fit, so be prepared to wait for a few minutes.

```
# random intercepts only
words.50.gam.int <- bam(f2 ~ word.ord + s(measurement.no, bs="cr") +
                            s(duration, bs="cr") +
                            ti(measurement.no, duration) +
                            s(measurement.no, by=word.ord, bs="cr") +
                            s(traj, bs="re"),
                        data=words.50, method="fREML")
# random intercepts + slopes
words.50.gam.slope <- bam(f2 ~ word.ord + s(measurement.no, bs="cr") +
                              s(duration, bs="cr") +
                              ti(measurement.no, duration) +
                              s(measurement.no, by=word.ord, bs="cr") +
                              s(traj, bs="re") +
                              s(traj, measurement.no, bs="re"),
                          data=words.50, method="fREML")
# random smooths
words.50.gam.smooth <- bam(f2 ~ word.ord + s(measurement.no, bs="cr") +
                             s(duration, bs="cr") +
                             ti(measurement.no, duration) +
                             s(measurement.no, by=word.ord, bs="cr") +
                             s(measurement.no, traj, bs="fs", xt="cr", m=1, k=5),
                           data=words.50, method="fREML")
```

Random intercepts are coded by including a smooth over the grouping variable with the smoothing class specified as `bs="re"`. This is really a technicality: `bam()` is able to use the mathematics of smooths to estimate various random structures, and this is reflected in its syntax as well. Random slopes are coded by adding the slope variable (`duration`) after the grouping variable (`traj`) inside the smooth, and keeping the smoothing class as `bs="re"`. This is a bit confusing, since random smooths have the opposite syntax: here, the continuous variable comes first, followed by the grouping variable. Let's repeat this below just to be on the safe side:

- random slopes: grouping factor, continuous variable
- random smooths: continuous variable, grouping factor

For random smooths, we also have to change the smoothing class to `bs="fs"`, which is short for 'factor smooth interactions'. The `m=1` specification is recommended in several papers (e.g. Baayen et al. 2016) for random smooths; what it does is slightly change the way the smoothing penalty is estimated (the default value is 2). The `xt="cr"` option sets the smooth class for the individual random smooths to cubic regression splines – this may not be necessary, but my impression was that cubic regression splines did a better job at capturing the current vowel trajectories. Finally, `k=5` sets a relatively low upper limit on the wiggliness of the individual random smooths. Although a higher number might result in a marginally better fit, this model already takes a long time to fit, and increasing `k` for random smooths can drastically increase the amount of resources (memory and time) needed to fit GAMMs.

The estimation method for the three models above is set to `method="fREML"`. We need restricted maximum likelihood estimation in this case since we want to compare models with the same fixed effects, but different random effects. In principle, we could also use `method="REML"`, but fREML ('fast REML') is much faster and tends to yield essentially the same results. If the comparison was between models with different fixed

effects but the same random effects (the typical case), the models would have to be estimated using `method="ML"` (see also Zuur et al. 2009; unfortunately, there is no 'fast ML', so models fitted with ML can take a long time to converge).

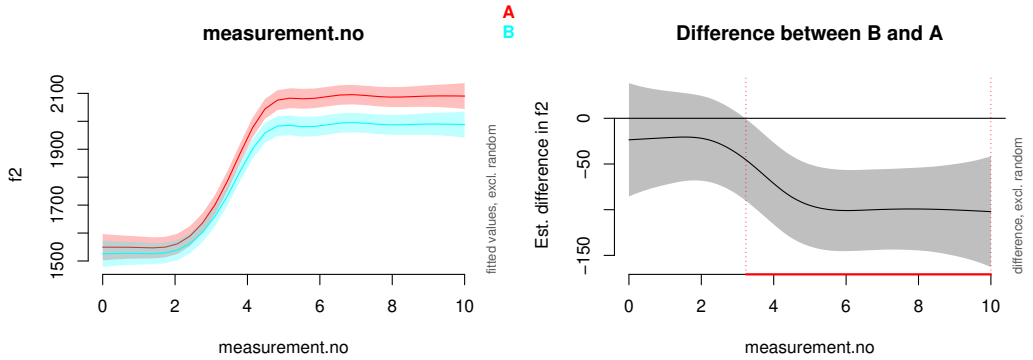
In order to find out which model fits the data best, we'll use a statistic called AIC (Akaike Information Criterion). This is necessary since the models are not all properly nested within each other. AIC is a combination of two quantities: how surprising the data are given our fitted model (the lower this number, the better the fit) and how many parameters are used in the model. That is, AIC penalises both bad model fits and unnecessary model complexity. When comparing two models, the one with a lower AIC should be preferred (AIC comparisons are slightly more complicated, but we will go with this simple heuristic for the current case). Here are the AIC values for the three models above:

```
AIC(words.50.gam.int, words.50.gam.slope, words.50.gam.smooth)

##                              df      AIC
## words.50.gam.int      70.02953 6173.767
## words.50.gam.slope   117.16199 5634.792
## words.50.gam.smooth 212.68868 5269.876
```

Based on these values, the model with random smooths is a clear winner: the added model complexity is more than compensated for by the improvement in model fit.

Let's recreate the model prediction plots that we previously generated for the simple model without random smooths. The plots below show predicted trajectories for words *A* and *B* (left), and a difference smooth for *B* − *A*.

```
plot_smooth(words.50.gam.smooth, view="measurement.no", plot_all="word.ord",
            rug=F, rm.ranef=T)
plot_diff(words.50.gam.smooth, view="measurement.no",
          comp=list(word.ord=c("B","A")), rm.ranef=T)
```



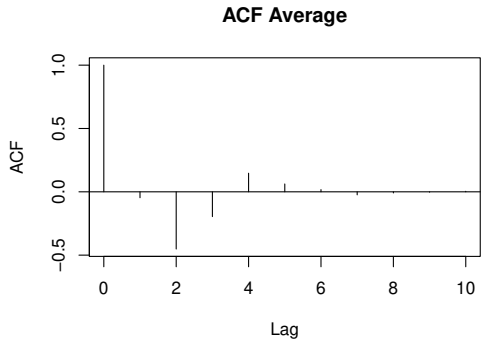
This is very similar to what we did for the simple model, but both functions include an additional option: `rm.ranef=T`. This option ensures that the predictions are shown for words *A* vs. *B* in general. If this option was left out, the predictions would relate to a given value of the grouping factor for the random effect, that is, `traj`. In other words, the plot would show predictions for a specific trajectory. But instead of just plotting that single trajectory, R would attempt to figure out what that trajectory would look like if it belonged to word *A* and what it would look like if it belonged to word *B*. Since a given

trajectory can only belong to one word, this can lead to unreliable confidence intervals and results that are difficult to interpret.

As expected, the confidence intervals for the model with random smooths are much wider than they are for the simple model (cf. the earlier graphs). Moreover, the shape of the difference smooth changes considerably between the two models: it looks more non-linear for the current model, and the confidence interval includes 0 until about `measurement.no` 3. Since the initial sections of the underlying curves for the two words are identical, this is a clear improvement on the previous model.

Can the added by-trajectory random smooths deal with the autocorrelation issue in the model? Below is the residual autocorrelation plot for the updated model.

```
acf_resid(words.50.gam.smooth, split_pred=c("traj"))
```



Since this model includes `traj` as a predictor, a different function can be used (though `acf_plot()` would also work fine): `acf_resid()` from `itsadug`. This function is a bit simpler in that it automatically extracts the residuals of the model and it's sufficient to specify the name of the predictor identifying the trajectories (using the `split_pred` option). The plot suggests that the autocorrelation at lag 1 is completely gone, although a certain amount of negative autocorrelation is introduced at lag 2. This is likely because of the fact that the initial and final sections of the raw trajectories are completely straight, but the fitted trajectories are actually a bit wavy. However, this negative autocorrelation in the residuals does not seem to lead to increased false positive rates in type I error simulations (at least not for this type of trajectory). This may be a consequence of the fact that there is very little variance left in the data after the addition of the by-trajectory random smooths.

The second option for reducing autocorrelation in the residuals is to use an autoregressive error model. There are many different kinds of autoregressive models, and `bam()` only supports the simplest of these: the so-called AR1 error model. An AR1 error model estimates the model parameters under the assumption that the errors[9] for neighbouring observations in a time series are correlated: for instance, the error at `measurement.no` 4 is partly determined by the error at `measurement.no` 3. The AR1 model assumes that such correlations only exist between immediately adjacent points

[9] The notion of 'error' is closely related to that of 'residual': errors are the deviations of the actual observations from the true underlying quantitites, while residuals are the deviations of the actual observations from the model predictions. Modelling correlations between errors has the practical effect of removing correlations in the residuals (assuming that we choose the right error model).

in the time series. An AR2 model would assume that the error at `measurement.no` 4 depends both on `measurement.no`'s 2 and 3.

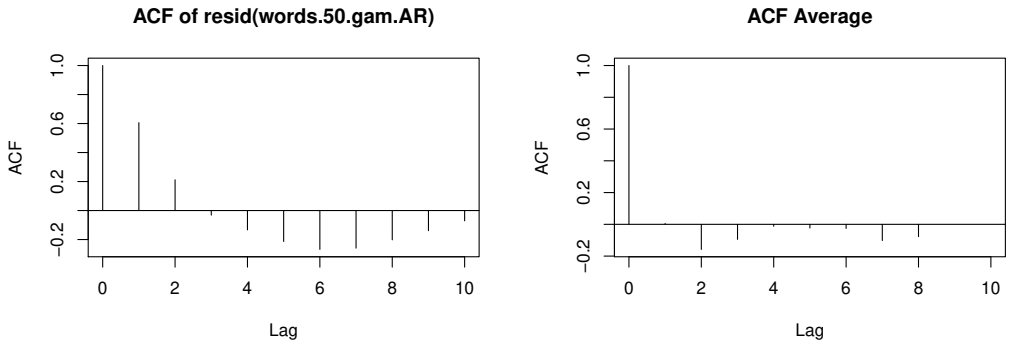
There are three things that need to be done before adding an AR1 model to a GAMM. First, the data set needs to be set up so that the order of observations in the data frame reflects their order in the time series (this is already how the current data frame is formatted). Second, we need to mark the starting point of each time series in the data set (i.e. `measurement.no` 0 for each trajectory) using a separate column – otherwise `bam()` wouldn't be able to determine which adjacent points in the data frame actually belong to the same time series, and which of them span two time series that just happen to be next to each other in the data set (e.g. the last `measurement.no` from one trajectory and the first `measurement.no` from the next trajectory). Third, `bam()` cannot estimate the degree of correlation between the errors, so that has to be specified manually. We'll use a rough estimate from the original model. Here's the code for fitting the model:

```
# 1) data frame is already ordered correctly
# 2) marking starting points of each trajectory

words.50$start.event <- words.50$measurement.no == 0

# 3) getting a rough estimate of the correlation between adjacent errors

r1 <- start_value_rho(words.50.gam.dur)

# fitting the model

words.50.gam.AR <- bam(f2 ~ word.ord + s(measurement.no, bs="cr") +
                           s(duration, bs="cr") +
                           ti(measurement.no, duration) +
                           s(measurement.no, by=word.ord, bs="cr"),
                       data=words.50, method="fREML",
                       rho=r1, AR.start=words.50$start.event)
```

The first line of code adds an indicator column to the data frame that has the value `TRUE` marking the beginning of each trajectory. The function `start_value_rho()` is from the package `itsadug`. It takes a GAMM without an autoregressive error model and returns the residual autocorrelation at lag 1. Note that this value should be treated as a rough estimate, and other values may, in fact, do a better job at reducing the autocorrelation in the residuals (see Baayen et al. 2016 for further discussion). The autoregressive model is added to the GAMM by setting the `rho` argument to the estimated correlation parameter and specifying the starting points of the time series using the `AR.start` argument.

Two plots are shown below, which are both generated using residual autocorrelation plotting functions.

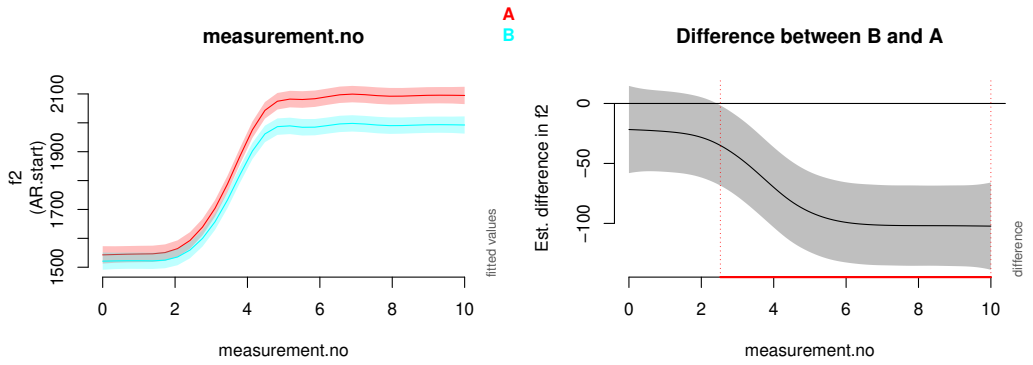
```
acf_plot(resid(words.50.gam.AR), split_by=list(words.50$traj))
acf_resid(words.50.gam.AR, split_pred="AR.start")
```



As should be clear from the syntax, both of these plots actually show the residuals from our updated model. However, the one on the left seems to suggest that the residual autocorrelation has not been affected by the AR1 model, while the one on the right suggests the opposite, that is, the AR1 model has been successful at reducing autocorrelation. This is because the plot on the left shows the raw residuals (which don't take the fitted AR model into account), while the model on the right shows the normalised residuals (which do). When an AR model is used, the plot generated using the raw residuals is misleading, so the plot on the right should be used. Note that this is created by the `acf_resid()` function, which works with the normalised residuals by default (but needs to be told that the data was split into separate time series, which were identified by the `AR.start` argument of the `bam()` function).

The plots below show the model predictions for words *A* and *B*, and the difference smooth.

```
plot_smooth(words.50.gam.AR, view="measurement.no", plot_all="word.ord", rug=F)
plot_diff(words.50.gam.AR, view="measurement.no",
          comp=list(word.ord=c("B","A")))
```



These plots look quite similar to the ones from the model including random smooths, though the confidence intervals are slightly narrower and the shapes of the fitted trajectories are subtly different.

So which method should be used: random smooths by trajectory or an AR1 model? In the current case, both of them seem to work fine and lead to very similar conclusions. However, that may not be the case for all data sets. For instance, Baayen et al. (2016) discuss a case where both random smooths and an AR model seem to be necessary in order to appropriately account for all dependencies within a data set. In deciding

between approaches, it is important to look at residual autocorrelation plots and also to check how much variance is actually left in the residuals.

There is, however, one practical consideration that makes the AR1 approach slightly more attractive. Adding an AR1 model to a GAMM is computationally inexpensive, while adding large numbers of random smooths can be very time- and memory-consuming. This may not be a problem for small data sets, but the computational cost of random smooths can be prohibitive for models with large numbers of trajectories.

What if we still want to fit random smooths to a large data set? The function `bam()` has an optional argument `discrete`, which speeds up computation substantially when set to `TRUE`. Moreover, when `discrete=TRUE`, a GAMM can be fitted using multiple processor cores. The number of processor cores used in the computation is specified by the `nthreads` argument. Note that this technique only works with the fREML fitting method, so evaluating significance through model comparison may not be possible.

4 Analysing data from Stuart-Smith et al. (2015)

We've applied GAMMs to various simulated data sets, but we haven't yet looked at any real data. In this section, we will use the methods introduced above to analyse a subset of the data presented in Stuart-Smith et al. (2015). This data set contains dynamic F3 measurements of word-final /r/ in spontaneous recordings of Glaswegian. The speakers are older males recorded at four different time points between 1970 and 2000. The data set is a fairly typical example of dynamic speech data. It is thoroughly unbalanced with varying numbers of tokens across speakers, decades, words and environments. Moreover, it likely shows the influence of a large number of different variables, though only a few of these are of interest for our present purposes. It is also larger than the toy data sets that we've worked with so far: the subset that we will look at contains 420 individual trajectories.

The data set consists of F3 trajectories measured at 11 evenly spaced points. The trajectories include both /r/ and the preceding vowel. The data come from four sets of three speakers recorded in the 1970s, the 1980s, the 1990s and the 2000s (i.e. 12 speakers altogether). The following variables will be used in the analysis:

- `measurement.no`: see above
- `duration`: overall duration of the vowel + /r/ sequence
- `decade`: decade of recording coded as a continuous variable
- `stress`: whether the previous vowel is an unstressed schwa or a full vowel
- `traj`: a grouping variable by trajectory
- `speaker`: a grouping variable by speaker

The main question that we'll try to answer is whether the acoustic characteristics of final /r/ have changed over time. That is, we'll be looking for an effect of `decade` on the F3 trajectories. However, there are a few complicating factors. First, the trajectories vary quite substantially in terms of their duration, and this is likely to affect their shapes: we might expect to see flatter trajectories for shorter vowel + /r/ sequences. Second, the

vowels in the vowel + /r/ sequences are not always the same, but their distribution is unbalanced: about 65% of all the vowels are schwas. Since the quality of the vowel may affect the F3 trajectory (though F3 is not expected to vary as much as F1 or F2 would), it is important to bring this factor into the analysis. To keep things simple, the quality of the vowel is encoded by the `stress` variable, which splits the data set according to whether the vowel is an unstressed schwa or a full vowel. We may also expect that stressed vs. unstressed vowel + /r/ sequences will change differently, though we have no clear prediction about what such a difference would look like.

Before we start analysing the data, there is one further important point that should be discussed. Although the analysis presented here will be shown in a relatively streamlined form, the model fitting procedure was actually preceded by a detailed exploration of the data through various plots. These plots won't be shown here, but many of the analytical decisions below are actually based on observations that emerged from this exploratory analysis. This type of data exploration is absolutely crucial, and there is almost no point in starting to fit GAMMs until we get a sense of the range of variation in the trajectories that we are trying to model.

To avoid starting with a very complex model, let's first simply try to capture the effect of `decade` on the trajectories. Since `decade` only across but not within speakers, it's essential to include `speaker` as a random smooth in the model. Otherwise the estimated effect of `decade` will be based on the assumption that there are 420 independent data points where, in reality, there are really only 12. We'll also record and display the amount of time it takes to fit the model using the `system.time()` function. This function displays three timing values. The last one of these shows how long it takes to fit the model overall; the other two are less relevant.

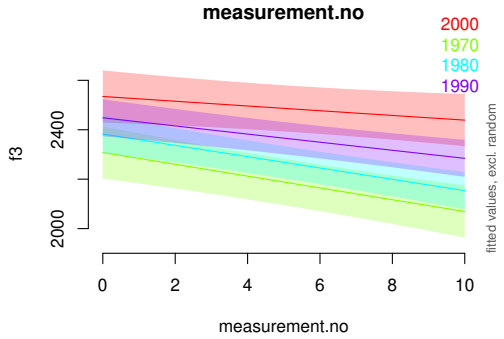
```
gl.r <- read.csv("glasgow_r.csv")
gl.r.gamm.simple.t <- system.time(
  gl.r.gamm.simple <- bam(f3 ~ s(measurement.no) +
                              s(decade, k=4) +
                              ti(measurement.no, decade, k=c(10,4)) +
                              s(measurement.no, speaker, bs="fs", m=1, k=4),
                          dat=gl.r, method="ML")
)
gl.r.gamm.simple.t

##    user  system elapsed
##   2.508   0.136   2.678

summary.coefs(gl.r.gamm.simple)

## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2327.37      27.48   84.71   <2e-16 ***
##
## Approximate significance of smooth terms:
##                               edf Ref.df      F  p-value
## s(measurement.no)           1.012  1.018 43.760 3.23e-11 ***
## s(decade)                   1.242  1.246 11.620 0.000181 ***
## ti(measurement.no,decade)   1.673  1.750  3.125 0.090800 .
## s(measurement.no,speaker) 30.990 46.000 19.176  < 2e-16 ***
```

There are a few things to note here. `k` cannot be set higher than 4 for `s(decade)`,

since `decade` only has 4 values. This also holds for the `ti()` interaction term, where `k` needs to be set separately for the two main terms. `k` is also set to 4 for the by-speaker random smooths, partly because the raw data don't show that much wiggliness and partly because we will eventually want to include some further random smooths, so it's worth keeping things reasonably simple.

Here's a plot illustrating the effect of `decade` on the trajectories:[10]

```
source("gamm_hacks.r")
plot_smooth.cont(gl.r.gamm.simple, view="measurement.no", plot_all.c="decade",
                 rug=F, rm.ranef=T)
```



We won't run model comparisons for this model, as a few things are still missing, but `decade` does seem to have a significant effect on average F3 and possibly even the slope of the trajectories. Interestingly, the predicted trajectories come out as completely straight (this can also be read off the model summary, where the EDF for all three fixed smooth terms is close to 1). This is an example of oversmoothing, which often happens when residual patterns aren't appropriately accounted for.

Next, we'll add two further variables: `duration` and `stress`. These two variables actually show a slight correlation, as unstressed schwas are shorter on average than full vowels, but there is sufficient overlap between the `duration` values for the two `stress` groups to estimate these effects separately. We'll add a main smooth term for `duration` as well as a `ti()` interaction with `measurement.no`. The predictor `stress` is included as a parametric term and a difference smooth (using `by=stress`).

```
gl.r$stress <- as.ordered(gl.r$stress)
contrasts(gl.r$stress) <- "contr.treatment"
gl.r.gamm.covs.t <- system.time(
  gl.r.gamm.covs <- bam(f3 ~ stress +
                          s(measurement.no) + s(measurement.no, by=stress) +
                          s(duration) + ti(measurement.no, duration) +
                          s(decade, k=4) +
                          ti(measurement.no, decade, k=c(10,4)) +
                          s(measurement.no, speaker, bs="fs", m=1, k=4),
                        dat=gl.r, method="ML")
)
gl.r.gamm.covs.t
```

[10] A slightly modified version of `plot_smooth()` is used to keep things simple. This function can be loaded by sourcing the file `gamm_hacks.r`.

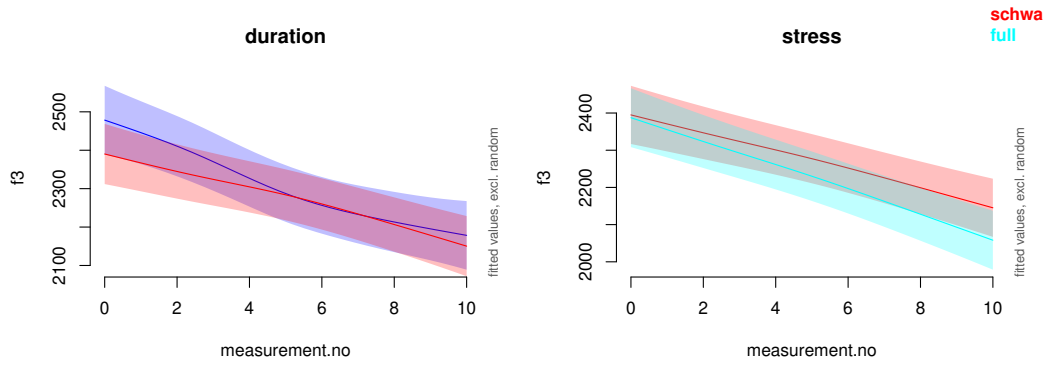
```
##    user  system elapsed
##   2.728   0.105   2.851
```

The inclusion of `stress` raises an additional problem: it is possible that the effect of stress varies across subjects. In a linear mixed effects model, this issue would be dealt with through random slopes (e.g. one could include by-subject random slopes for the interaction between `measurement.no` and `stress` and the corresponding main terms). Although such solutions could also be explored for the current case, we'll follow a random smooth-based technique from Wieling et al. (2016) instead. We will fit separate smooths for each speaker at each level of `stress`, which will allow us to capture speaker-specific trends in the `stress` effect. This can be achieved by using a combined `speaker` × `stress` variable as the grouping factor for the random smooths. The model is shown below.

```
# creating combined grouping variable
gl.r$speakerStress <- interaction(gl.r$speaker, gl.r$stress)

gl.r.gamm.covs.2.t <- system.time(
  gl.r.gamm.covs.2 <- bam(f3 ~ stress +
                            s(measurement.no) + s(measurement.no, by=stress) +
                            s(duration) + ti(measurement.no, duration) +
                            s(decade, k=4) +
                            ti(measurement.no, decade, k=c(10,4)) +
                            s(measurement.no, speakerStress, bs="fs", m=1, k=4),
                          dat=gl.r, method="ML")
)
gl.r.gamm.covs.2.t

##    user  system elapsed
##   6.802   0.346   7.226
```

Note that the function `interaction()` simply combines `speaker` and `stress` in a single variable. To save space, the model summary is not shown, but here's a graphical summary of the effects of `duration` and `stress`.

```
plot_smooth(gl.r.gamm.covs.2, view="measurement.no", cond=list(duration=0.3),
            rug=F, rm.ranef=T, col="blue", main="duration")
plot_smooth(gl.r.gamm.covs.2, view="measurement.no", cond=list(duration=0.1),
            rug=F, rm.ranef=T, col="red", add=T)
plot_smooth(gl.r.gamm.covs.2, view="measurement.no", plot_all="stress",
            rug=F, rm.ranef=T, main="stress")
```



The F3 trajectories start lower for shorter /Vr/ sequences and are a bit flatter (the red line represents a short trajectory). This suggests that short vowels are more strongly coarticulated with the following /r/. Moreover, there is a more pronounced dip near the /r/ part of the sequence for sequences with full vowels. Again, the unstressed sequence shows a flatter trajectory. One possible interpretation is that /r/ is more likely to be weakened or deleted after unstressed vowels.

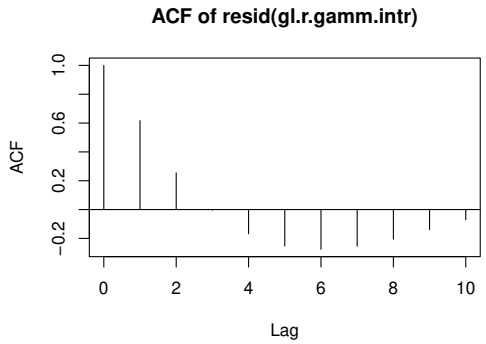
Finally, let's see whether `stress` interacts with the effect of `decade`. Two further `by` terms need to be added: one for the `decade` main term and another one for the `ti()` interaction between `measurement.no` and `decade`. Only the added terms are shown in the code chunk below.

```
gl.r.gamm.intr <- bam(f3 ~ stress +
                          ...
                          s(decade, k=4, by=stress) +
                          ti(measurement.no, decade, k=c(10,4), by=stress),
                      dat=gl.r, method="ML")

##   user  system elapsed
## 12.390   0.411  12.818
```

The summary for this model (not shown above) suggests that `stress` and `decade` do not interact significantly.

How does this model fare with respect to residual autocorrelation? Since we haven't yet included by-trajectory random smooths and/or an autoregressive error model, there will likely be some autocorrelation in the residuals. This is confirmed by the residual autocorrelation plot below:

```
acf_plot(resid(gl.r.gamm.intr), split_by=list(gl.r$traj))
```



The residual patterns look very similar to those from the simpler models of the previous section, and are due to the fact that the grouping structure in the data is ignored. This likely has an influence on the estimated effects of `stress` and `duration`, which should really be estimated from differences across trajectories, not individual data points. Since the model does not include by-trajectory random smooths or an AR error model, it is not aware of dependencies among data points from the same trajectories.

Let's first try fitting the model using by-trajectory random smooths. We'll change the estimation method to fREML as ML is simply too slow and memory-intensive. This, of course, means that we cannot check for the significance of fixed terms using model

comparison. Note also that the value of `discrete` is set to TRUE to speed up computation. If your machine has more than one processor core, you can try setting `nthreads` to a value higher than one, which may speed up processing even further.

```
gl.r.gamm.traj.t <- system.time(
  gl.r.gamm.traj <- bam(f3 ~ stress +
                          s(measurement.no) + s(measurement.no, by=stress) +
                          s(duration) + ti(measurement.no, duration) +
                          s(decade, k=4) + s(decade, k=4, by=stress) +
                          ti(measurement.no, decade, k=c(10,4)) +
                          ti(measurement.no, decade, k=c(10,4), by=stress) +
                          s(measurement.no, speakerStress, bs="fs", m=1, k=4) +
                          s(measurement.no, traj, bs="fs", m=1, k=4),
                        dat=gl.r, method="fREML", discrete=T)
  )
gl.r.gamm.traj.t

##    user  system elapsed
## 220.948  19.134 243.193

summary.coefs(gl.r.gamm.traj, digits=3)

## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2303.8       31.7   72.72   <2e-16 ***
## stressschwa     42.2       43.6    0.97     0.33
##
## Approximate significance of smooth terms:
##                                          edf  Ref.df      F p-value
## s(measurement.no)                       6.43    7.35   9.65 2.7e-12 ***
## s(measurement.no):stressschwa           5.24    6.19   2.74 0.01106 *
## s(duration)                             1.87    1.87   0.94 0.29569
## ti(measurement.no,duration)             5.59    6.35   3.60 0.00120 **
## s(decade)                               1.93    1.93  10.44 0.00022 ***
## s(decade):stressschwa                   1.00    1.00   0.24 0.62743
## ti(decade,measurement.no)              11.57   14.12   2.01 0.01198 *
## ti(decade,measurement.no):stressschwa   6.40    8.34   0.34 0.95583
## s(measurement.no,speakerStress)        45.31   94.00   1.54 < 2e-16 ***
## s(measurement.no,traj)               1536.71 1676.00 101.81 < 2e-16 ***
```

The same model fit with an AR1 error term is shown on the next page.

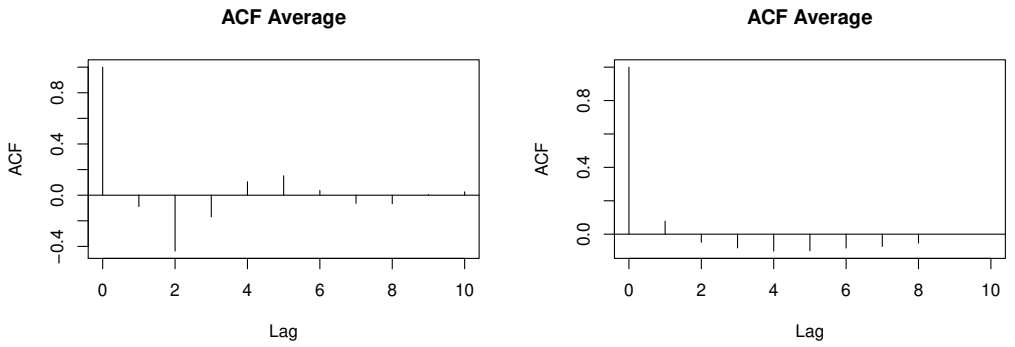
```
# marking start of trajectories
gl.r$start.event <- gl.r$measurement.no == 0
# getting rough estimate of autocorrelation parameter
gl.autocorr <- start_value_rho(gl.r.gamm.intr)

# fit model
gl.r.gamm.AR.t <- system.time(
  gl.r.gamm.AR <- bam(f3 ~ stress +
                        s(measurement.no) + s(measurement.no, by=stress) +
                        s(duration) + ti(measurement.no, duration) +
                        s(decade, k=4) + s(decade, k=4, by=stress) +
                        ti(measurement.no, decade, k=c(10,4)) +
                        ti(measurement.no, decade, k=c(10,4), by=stress) +
                        s(measurement.no, speakerStress, bs="fs", m=1, k=4),
                      dat=gl.r, method="ML",
                      AR.start=gl.r$start.event, rho=gl.autocorr)
  )
gl.r.gamm.AR.t

##    user  system elapsed
##  19.309   0.868  20.683

summary.coefs(gl.r.gamm.AR, digits=3)

## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2295.4       28.6   80.16   <2e-16 ***
## stressschwa     47.3       39.9    1.19     0.24
##
## Approximate significance of smooth terms:
##                                        edf Ref.df     F p-value
## s(measurement.no)                     4.78   6.07  9.84 6.9e-11 ***
## s(measurement.no):stressschwa         1.01   1.01  1.38   0.241
## s(duration)                           2.35   2.99  2.94   0.035 *
## ti(measurement.no,duration)           6.26   8.10  4.61 1.2e-05 ***
## s(decade)                             1.50   1.52 11.49 6.7e-05 ***
## s(decade):stressschwa                 1.00   1.00  0.37   0.542
## ti(measurement.no,decade)             2.45   3.13  1.61   0.190
## ti(measurement.no,decade):stressschwa 1.01   1.01  0.04   0.846
## s(measurement.no,speakerStress)      67.45  92.00  6.39 < 2e-16 ***
```

First of all, note that the AR model converges about 10 times faster than the model with random smooths, even though it is fitted with ML, which is generally a bit slower (ML was chosen so that we can perform model comparison later). There are also differences in significance across the two models. For some variables, the model with random smooths is more conservative (`duration`); for others, it is less conservative (`stress`). Such differences only really occur in cases where the *p*-values are relatively close to 0.05. The fixed effects that are significant in both models are `s(decade)` and `ti(measurement.no,duration)`.

The residual autocorrelation plots for the two models are shown below. The results are very similar to those from the previous sections: the model with random smooths by trajectory introduces an artefactual negative autocorrelation at lag 2 (though, as before, there is little residual variance left in the data, so this may be less problematic than it appears from the autocorrelation plot), while the model with an autoregressive error

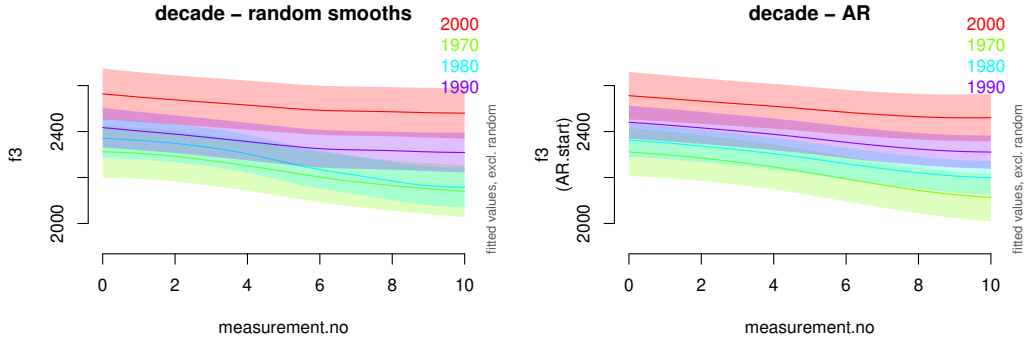
model gets rid of most of the autocorrelation without introducing any artefacts.

```
# autocorrelation plots
acf_resid(gl.r.gamm.traj, split_pred="traj")
acf_resid(gl.r.gamm.AR, split_pred="AR.start")
```



Let's compare the predictions of the models. The plots below show the estimated `decade` effect (for schwa) from the two models.

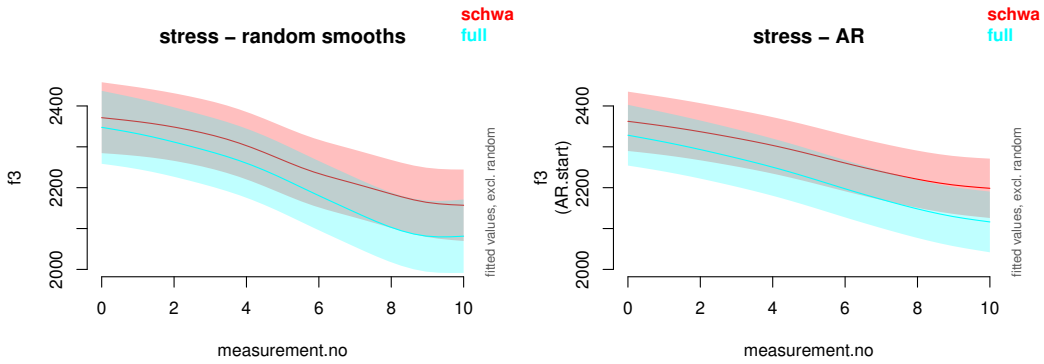
```
plot_smooth.cont(gl.r.gamm.traj, view="measurement.no", plot_all.c="decade",
                 cond=list(stress="schwa"), rug=F, rm.ranef=T,
                 main="decade - random smooths", ylim=c(1900,2700))
plot_smooth.cont(gl.r.gamm.AR, view="measurement.no", plot_all.c="decade",
                 cond=list(stress="schwa"), rug=F, rm.ranef=T,
                 main="decade - AR", ylim=c(1900,2700))
```



Though the two plots are very similar, the estimated trajectory shapes are slightly different, especially for the 1980s speakers (this is likely a reflection of the fact that the smooth interaction between `decade` and `measurement.no` is significant in the model with random smooths, but not in the one with an AR1 model). The trajectories estimated by the GAMM with the AR model are also generally a bit smoother.

Here are the predictions for stressed vowel + /r/ *vs.* unstressed vowel + /r/ sequences.

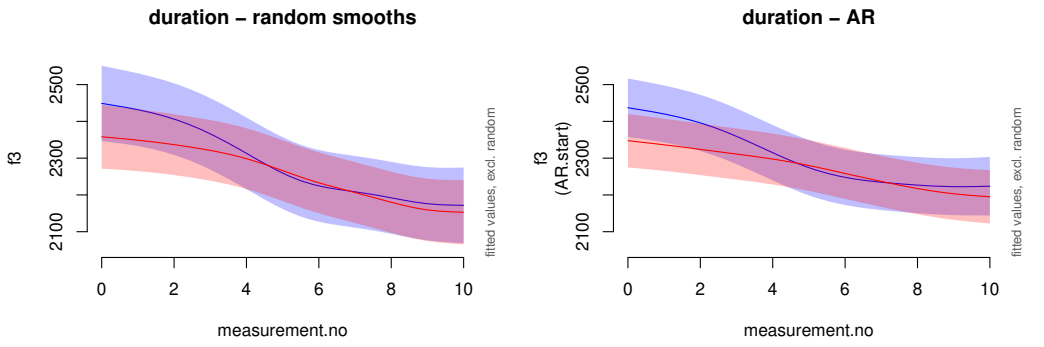
```
plot_smooth(gl.r.gamm.traj, view="measurement.no", plot_all="stress",
            rug=F, rm.ranef=T, main="stress - random smooths", ylim=c(2000,2450))
plot_smooth(gl.r.gamm.AR, view="measurement.no", plot_all="stress",
            rug=F, rm.ranef=T, main="stress - AR", ylim=c(2000,2450))
```



Again, the estimated trajectories look similar, but the AR ones are somewhat smoother (and since they run almost in parallel, the shape difference that is significant for the random smooth model is not significant here – see the model summaries).

The predictions for V + /r/ sequences with short *vs.* long durations:

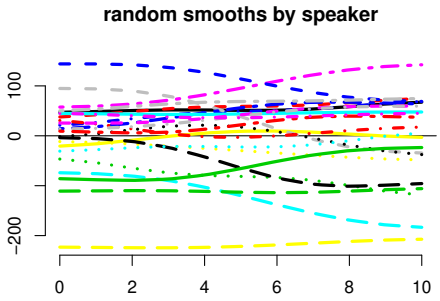
```
plot_smooth(gl.r.gamm.traj, view="measurement.no", cond=list(duration=0.3),
            rug=F, rm.ranef=T, col="blue", main="duration - random smooths",
            ylim=c(2050,2550))
plot_smooth(gl.r.gamm.traj, view="measurement.no", cond=list(duration=0.1),
            rug=F, rm.ranef=T, col="red", add=T)
plot_smooth(gl.r.gamm.AR, view="measurement.no", cond=list(duration=0.3),
            rug=F, rm.ranef=T, col="blue", main="duration - AR",
            ylim=c(2050,2550))
plot_smooth(gl.r.gamm.AR, view="measurement.no", cond=list(duration=0.1),
            rug=F, rm.ranef=T, col="red", add=T)
```



Though the difference in smoothness seen for the previous graphs is also observed here, the general conclusions from the two models with respect to duration are very similar.

The plot below shows the estimated random smooths for different speakers from the model with random smooths. In order to get the graph to display the correct set of random smooths, we need to specify it using a number, which actually corresponds to its position in the smooth part of the model summary (i.e. it's on the 9th line in the smooth summary).

```
inspect_random(gl.r.gamm.traj, select=9, lwd=3,
               main="random smooths by speaker")
```



Finally, we'll perform a model comparison to check whether the effect of decade is significant. The comparison is between the full GAMM with an AR model and a nested model that excludes *all* terms with `decade`. The comparison is shown below:

```
gl.r.gamm.AR.0 <- bam(f3 ~ stress +
                          s(measurement.no) + s(measurement.no, by=stress) +
                          s(duration) + ti(measurement.no, duration) +
                          s(measurement.no, speakerStress, bs="fs", m=1, k=4),
                      dat=gl.r, method="ML",
                      AR.start=gl.r$start.event, rho=gl.autocorr)
compareML(gl.r.gamm.AR, gl.r.gamm.AR.0, print.output=F)$table

##            Model    Score Edf  Chisq     Df p.value Sig.
## 1 gl.r.gamm.AR.0 26391.20  13
## 2   gl.r.gamm.AR 26377.75  23 13.450 10.000   0.003  **
```

The difference between the models is significant, indicating that there is indeed a change in F3 as a function of time. This concludes our analysis of the Glasgow /r/ data set.

One remaining question to answer is whether we should prefer the GAMM with random smooths by trajectory or the one with an AR model – or should we perhaps combine them? In this case, combining the two methods is probably not a good idea: though the random smooths introduce an artefactual correlation, an AR1 model wouldn't be able to deal with this issue appropriately, as it is at lag 2, not lag 1. There are no such artefacts in the AR1 version of the model, which tips the balance in favour of the latter model. Moreover, the AR1 version can be fitted using ML, which in turn makes model comparisons possible. In contrast, the version of the model with random smooths is too complex to be fitted using ML, so model comparison based on fixed effects is not an option. Finally, the AR1 version can be fitted in a much shorter time. This may not be a knock-down argument in the current case, but it is a very important consideration for data sets with a larger number of trajectories. Therefore, although the overall conclusions from the two models are similar, a GAMM with an AR1 error model seems preferable to a GAMM with random smooths by trajectory for the Glasgow /r/ data set.

It should be noted that while these arguments may hold for dynamic formant data of the type presented here, there is no one-size-fits-all solution for this issue, and the choice between the two options should be guided by careful consideration of the data and

model criticism (e.g. inspection of residual autocorrelation plots). There may be also be cases where both types of structures are needed to account for patterns in the residuals. For more information on these issues, I strongly recommend Baayen et al. (2016) and Baayen et al. (2017): both papers offer plenty of advice on using by-trajectory random smooths and autoregressive error models, and also illustrate their use through a wide variety of linguistic examples.

5 Final comments

The goal of this paper was to provide an introduction to GAMMs in the context of dynamic speech analysis. We have discussed a range of theoretical concepts and gone through two example data sets, covering many of the questions and issues that come up when working with these models. However, there are many more issues that I simply had to leave out in order to keep things short:

- checking modelling assumptions: normal distribution of residuals, homoscedasticity
- dealing with violations of these assumptions
- other smoother types such as adaptive smoothers and cyclic smoothers (might be appropriate for pitch contours)
- GAMMs for two-dimensional spatial data

5.1 Useful references

Below is a short (and incomplete) list of useful GAMM references. I consulted many of these while putting this introduction together.

- Wood (2006): The standard reference for GAMMs. It is an excellent book with a lot of detail, and can be very useful for finding out more about the methods and assumptions underlying GAMMs. Though many of the sections assume a strong background in mathematics, most of the example analyses and parts of the conceptual discussion are possible to follow without mathematical training.
- Baayen et al. (2016): A paper that focuses specifically on modelling autocorrelation using GAMMs, and presents analyses for three separate linguistic examples. It also includes some complicated GAMMs (even some with three-way interactions).
- Baayen et al. (2017): A paper that looks at typical patterns of temporal autocorrelation in data obtained from humans. Provides advice on handling autocorrelation as well as discussion of other modelling issues (e.g. the difference between exploratory and confirmatory analyses).
- Kelly (2014): An online tutorial. This one covers not just GAMMs, but also a range of other methods for dealing with non-linearity, and makes various comparisons across these methods. Note that this tutorial relies on the `gam` library, not `mgcv`.

- Simpson (2014): An interesting blog post that shows how to model seasonal data using GAMMs. It also explains basis functions in a clear and concise way.
- Simpson (2011): Another great blog post with a lot of useful detail on autocorrelation components. It also has an interesting section on generating confidence intervals for the derivative of a smooth function.
- van Rij (2015): A great online tutorial that covers many of the technical aspects of fitting GAMMs. It also includes autocorrelation and model comparisons.
- van Rij (2016): A brief R vignette that describes three different methods for significance testing using GAMMs. Note that some of the model comparisons are based on GAMMs fitted with fREML, which may lead to unreliable results (this is noted in the text as well).
- Winter & Wieling (2016): Discusses GAMMs in the context of language change and evolution and provides a clear and concise introduction to GAMMs and mixed models in general. It also covers a number of topics that are not discussed here, such as autocorrelation and logistic / Poisson GAMMs. It comes with thoroughly commented example code that is also available at `https://github.com/bodowinter/change_tutorial_materials`.
- Wieling (2017): Lecture slides for a five-day workshop on advanced regression methods. Lectures 3–5 provide a very detailed introduction to GAMMs with tons of useful examples illustrating ways of dealing with autocorrelation, non-normally distributed residuals and also a range of different types of data (including ERP and articulatory data).

References

Baayen, R. H. (2008). *Analyzing linguistic data: A practical introduction to statistics using R*. Cambridge: Cambridge University Press.

Baayen, R. H., van Rij, J., de Cat, C., & Wood, S. N. (2016). Autocorrelated errors in experimental data in the language sciences: Some solutions offered by generalized additive mixed models. arXiv preprint arXiv:1601.02043.

Baayen, R. H., Vasishth, S., Bates, D., & Kliegl, R. (2017). The cave of shadows. addressing the human factor with generalized additive mixed models. *Journal of Memory and Language*, 94, 206–234.

Barr, D. J., Levy, R., Scheepers, C., & Tily, H. J. (2013). Random effects structure for confirmatory hypothesis testing: Keep it maximal. *Journal of memory and language*, 68(3), 255–278.

Bates, D., Kliegl, R., Vasishth, S., & Baayen, H. (2015). Parsimonious mixed models. *arXiv preprint*, arXiv:1506.04967.

Bates, D., Maechler, M., & Bolker, B. (2011). *lme4: Linear mixed-effects models using S4 classes*. R package version 0.999375-42.

Gelman, A. & Hill, J. (2007). *Data analysis using regression and multilevel/hierarchical models*. Cambridge: Cambridge University Press.

Johnson, K. (2008). *Quantitative methods in linguistics*. Malden, MA & Oxford: Blackwell.

Kelly, R. (2014). Extending linear models: Non-linearity. https://rstudio-pubs-static.s3.amazonaws.com/24589_7552e489485b4c2790ea6634e1afd68d.html. Accessed on 23/01/2017.

R Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Simpson, G. (2011). Additive modelling and the hadcrut3v global mean temperature series. http://www.fromthebottomoftheheap.net/2011/06/12/additive-modelling-and-the-hadcrut3v-global-mean-temperature-series/. Accessed on 23/01/2017.

Simpson, G. (2014). Modelling seasonal data with GAMs. http://www.fromthebottomoftheheap.net/2014/05/09/modelling-seasonal-data-with-gam/. Accessed on 23/01/2017.

Sóskuthy, M. (2016). Generalised additive mixed modelling for dynamic formant analysis. Talk given at LabPhon 2015, Cornell University, Ithaca, NY.

Sóskuthy, M. (in prep). Significance testing in dynamic speech analysis using generalised additive mixed models. Unpublished manuscript.

Stuart-Smith, J., Lennon, R., Macdonald, R., Robertson, D., Sóskuthy, M., José, B., & Evers, L. (2015). A dynamic acoustic view of real-time change in word-final liquids in spontaneous glaswegian. *Proceedings of the 18th International Congress of Phonetic Sciences*, 10-14 August 2015, Glasgow.

van Rij, J. (2015). Overview GAMM analysis of time series data. http://www.sfs.uni-tuebingen.de/~jvanrij/Tutorial/GAMM.html. Accessed on 23/01/2017.

van Rij, J. (2016). Testing for significance. https://cran.r-project.org/web/packages/itsadug/vignettes/test.html. Accessed on 23/01/2017.

van Rij, J., Wieling, M., Baayen, R. H., & van Rijn, H. (2016). itsadug: Interpreting Time Series and Autocorrelated Data using GAMMs. R package version 2.2.

Venables, W. N. & Ripley, B. D. (2002). *Modern applied statistics with S-PLUS*. New York: Springer.

Wieling, M. (2017). Advanced regression for linguists [lecture slides]. http://www.let.rug.nl/~wieling/statscourse/. Accessed on 10/03/2017.

Wieling, M., Tomaschek, F., Arnold, D., Tiede, M., Bröker, F., Thiele, S., Wood, S. N., & Baayen, R. H. (2016). Investigating dialectal differences using articulography. *Journal of Phonetics*, 59, 122–143.

Winter, B. (2013). Linear models and linear mixed effects models in r with linguistic applications. *arXiv preprint*, arXiv:1308.5499.

Winter, B. & Wieling, M. (2016). How to analyze linguistic change using mixed models, Growth Curve Analysis and Generalized Additive Modeling. *Journal of Language Evolution*, 1(1), 7–18.

Wood, S. (2006). *Generalized additive models: an introduction with R*. Boca Raton: CRC Press.

Zuur, A., Ieno, E. N., Walker, N., Saveliev, A. A., & Smith, G. M. (2009). *Mixed effects models and extensions in ecology with R*. New York: Springer.

Appendix: R packages and functions for fitting GAMMs

There are a number of different functions that can be used for fitting GAMMs in R: `gam()`, `bam()`, `gamm()` (all three from the package `mgcv`) and `gamm4()` (from the package `gamm4`). These functions have slightly different strength and weaknesses. Some of the discussion below is fairly technical, and probably makes more sense to readers who already have a bit of experience with GAMMs. I have included it here as I think it may be useful for reference.

`gam()`/`bam()` are the best documented and most reliable tools for fitting GAMMs, and most existing tutorials focus on these functions. Although these are separate functions, they can be used in very similar ways. `bam()` is in many ways a more advanced version of `gam` that can be much faster than `gam()` and uses less memory, so it is the preferred option for large or complex data sets. `bam()` can also be run in parallel on multiple processor cores and includes an option for further performance gains (these can be accessed by setting `discrete=TRUE` and specifying the number of parallel threads using the `nthreads` option). `bam()` also allows the inclusion of a simple autoregressive error model of order 1 (AR1), which can capture some patterns of autocorrelation in the residuals. `gam()`/`bam()` are generally more versatile than `gamm()` and `gamm4()`, though the latter two can be a bit faster when the model contains random smooths with many levels. The package `itsadug` has been developed primarily with `gam()`/`bam()` in mind, and many of its functions do not work properly with models fitted using `gamm()` or `gamm4()`. Here is a brief list of the main features of `gam()` that are relevant for us (don't worry if some of these features are unclear at this point; many of them will be discussed soon):

- can fit all smooth types, including traditional random effects and random smooths
- can fit crossed random smooths (e.g. random smooths by words and by speakers at the same time)
- fully compatible with `itsadug`
- can fit smooth interactions (`te()` and `ti()`), where the main effects and interaction terms are separable
- possible to compare models using `anova()` / `compareML()`

- possible to model simple autocorrelation in the data when using `bam()` (but not `gam()`)
- can be run on multiple processor cores (`bam()`)

`gamm()` is superficially similar to `gam()/bam()`, but it relies on an external package (`nlme`) for estimating models. In practical terms, this means that `gamm()` can be faster than `gam()/bam()` when the model includes randoms smooths with many levels. In addition, `gamm()` differs from `gam()/bam()` in the following ways:

- cannot fit crossed random smooths
- only partly compatible with `itsadug`
- not possible to compare models using `anova()`
- can include complex models of autocorrelation
- can deal with heteroscedascity in the data by using variance components

`gamm4()` is in many ways similar to `gamm()`: it mostly uses the same syntax as `gam()/bam()`, but performs model fitting with the help of the `lme4` package. Like `gamm()`, `gamm4()` is good at fitting models with random smooths, and possibly even faster than `gamm()`. It also comes with its specific set of pros and cons compared to `gam()/bam()` and `gamm()`:

- can fit crossed random smooths
- only partly compatible with `itsadug`
- possible to compare models using `anova()`
- cannot deal with autocorrelation or heteroscedascity
- can only fit smooth interactions where the main effects and interaction terms are inseparable (so their significance cannot be evaluated separately)